Алан Купер
Психбольница в руках пациентов. Алан Купер об интерфейсах

Библиотека программиста (Питер) –

предоставлен правообладателем Издательский дом "Питер" 2018
ISBN 978-5-4461-0674-5

Аннотация

Все мы – безумцы, живущие в технологическом сумасшедшем доме, и создали этот безумный мир мы сами. Своими руками сотворили этот кошмар: интерфейсы, которые нас раздражают и утомляют глаза, устройства, которые приводят к болям в спине и в запястьях. Эта книга стала манифестом и до сих пор не потеряла актуальность. Дверь на свободу распахнута. Почему же мы не замечаем выхода? Об этом и рассказывает Алан Купер, объясняя разницу между интерфейсом и взаимодействием.

Эй, ребята, у вас тут полно обозленных клиентов. Вам есть что им ответить?

А. Купер

Психбольница в руках пациентов. Алан Купер об интерфейсах
или Почему высокие технологии сводят нас с ума и как восстановить душевное равновесие

Alan Cooper
THE INMATES ARE RUNNING THE ASYLUM:
Whe High-Tech Products Drive Us Crazy And How to Restore the Sanity

Серия «Библиотека программиста»

*** * ***

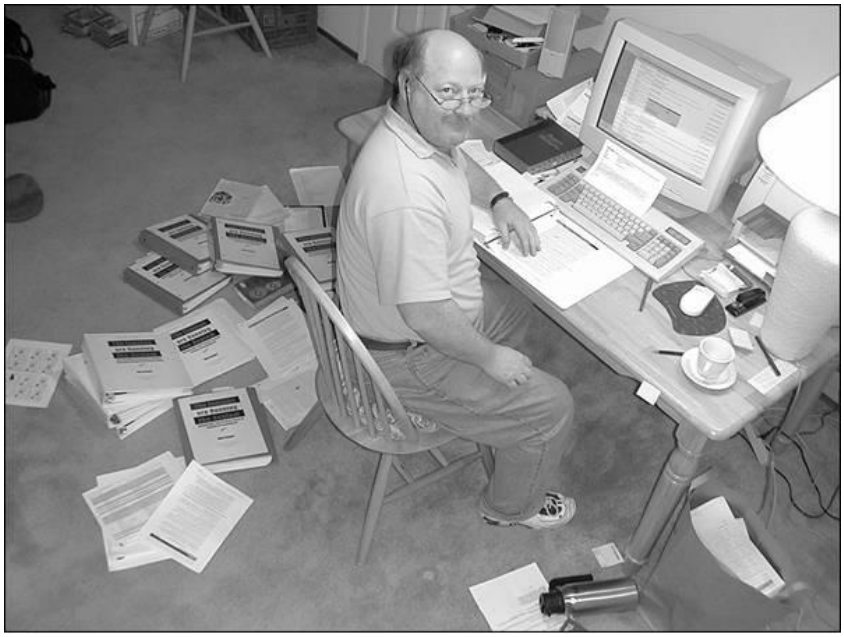
Сью, Скотту и Марти, с любовью

Благодарности

Этой книги не было бы без помощи и поддержки многих моих замечательных друзей и коллег. Некоторые из них, в частности, выполняли сложную работу со множеством требований – вычитывали и комментировали рукопись, иногда даже по несколько раз. Благодаря их комментариям я отвечал на нелегкие вопросы, предлагал новые темы, резюмировал свои убеждения, усмирял свой пыл и брал в узду свое возмущение. Еще лучше этой книге помогли стать Ким Гудвин (Kim Goodwin), Лейн Хэлли (Lane Halley), Келли Боуман (Kelly Bowman), Скотт Макгрегор (Scott McGregor), Дэвид Уэст (David West), Майк Нельсон (Mike Nelson), Марк Джирск (Mark Dziersk), Алан Карп (Alan Karp), Терри Суок (Terry Swack), Луи Вейцман (Louie Weitzman), Уэйн Гринвуд (Wayne Greenwood), Райан Ольшавски (Ryan Olshavsky), Джон Майер (John Meyer), Лайза Сондерс (Lisa Saunders), Винни Шоуз (Winnie Shows), Кевин Вандрайк (Kevin Wandryk), Глен Халстед (Glenn Halstead), Брайан О'Салливан (Bryan O'Sullivan), Чак Оуэн (Chuck Owen), Майк Суэйн (Mike Swaine), а также Скип Уолтер (Skip Walter). Я благодарен всем вам за ваше время, поддержку и мудрость. Особую ценность для меня также возымели советы и комментарии Джонатана Кормана (Jonathan Korman) – с их помощью я смог лучше выделить темы для книги. Я также должен поблагодарить всех невероятно талантливых и трудолюбивых сотрудников из *Cooper Interaction Design*, которые взяли мои обязанности на себя, пока я занимался подготовкой

книги. Особую благодарность выражаю директору по проектированию Уэйну Гринвуду (Wayne Greenwood), которому пришлось работать в жестких стрессовых условиях, сохраняя при этом высокое качество наших проектов и не утрачивая боевой дух.

Одним из самых интересных вызовов при создании этой книги стала задача подготовки иллюстраций. Чед Кубо (Chad Kubo), гениальный создатель иллюстраций, проделал невероятную работу по претворению моих смутных задумок в четкие и запоминающиеся изображения. Благодаря им книга заиграла другими красками. Но этих иллюстраций не было бы без интенсивного руководства художественной частью в исполнении Пенни Бейлес (Penny Bayless) и Дэвида Хейла (David Hale). Другие специалисты помогали с решением производственных задач. Я благодарю Брит Кацен (Brit Katzen) за выверку фактов и исследования, а также Майка Генри (Mike Henry) за редактуру текста.



Создание книги – это бизнес-проект, и за его успех я должен выразить благодарность моей команде подкованных в технологиях предпринимателей, во главе которой стоял мой агент Джим Левайн (Jim Levine), а саму команду составляли Глен Халстед (Glenn Halstead), Линн Боуман (Lynne Bowman), Келли Боуман (Kelly Bowman) и Сью Купер (Sue Cooper). Также поддержку по ходу всего проекта оказывало и издательство *Pearson* в лице Брэда Джоунса (Brad Jones), но самые большие благодарности я хотел бы высказать Крису Веббу (Chris Webb) за его целеустремленность, высочайшую концентрацию и тяжелый труд, без которых эта книга вообще бы не состоялась.

Для меня невероятно ценно внимание всех тех людей, что оказывали мне моральную поддержку, а также снабжали меня историями, советами и тратили на проект время. Огромное спасибо таким людям, как Дэниел Эпплман (Daniel Appleman), Тодд Баше (Todd Basche), Крис Байер (Chris Bauer), Джефф Безос (Jeff Bezos), Элис Блэр (Alice Blair), Мишель Борк (Michel Bourque), По Бронсон (Po Bronson), Стив Калде (Steve Calde), Дэвид Карлик (David Carlick), Джефф Карлик (Jeff Carlick), Кэрол Кристи (Carol Christie), Клэй Коллье (Clay Collier), Кендалл Косби (Kendall Cosby), Дэн Крэйн (Dan Crane), Роберт Кringели (Robert X. Cringely), Трой Дэниелс (Troy Daniels), Лайза Пауэрс (Lisa Powers), Филип Энглхарт (Philip Englehardt), Карен Ивенсен (Karen Evensen), Риджели Эверс (Ridgely Evers), Ройял Фаррос (Royal Farros), Пэт Флек (Pat Fleck), Дэвид Фор (David Fore), Эд Форман (Ed Forman), Эд Фридкин (Ed Fredkin), Жан-Луи Гассе (Jean-Louis Gassee), Джим Гэй (Jim Gay), Расс Голдин (Russ Goldin), Влад Горелик (Vlad Gorelik), Марсия Грегори (Marcia Gregory), Гаррент Грюнер (Garrett Gruener), Чак Хартледж (Chuck Hartledge), Тед Харвуд (Ted Harwood), Уилл Хирст (Will Hearst), Тамра Хизершоу-Харт (Tamra Heathershaw-Hart),

Джей-Ди Хильдебранд (JD Hildebrand), Лори Хиллз (Laurie Hills), Питер Хиршберг (Peter Hirshberg), Ларри Кили (Larry Keeley), Гэри Краткин (Gary Kratkin), Дебора Курата (Deborah Kurata), Том Лефлер (Tom Lafleur), Пол Лотон (Paul Laughton), Элен Леви (Ellen Levy), Стивен Лист (Steven List), Ти-Си Манган (TC Mangan), Дэвид Майстер (David Maister), Роберт Мэй (Robert May), Дон Маккини (Don McKinney), Кэтрин Медоуз (Kathryn Meadows), Лайза Митчелл (Lisa Mitchell), Джеффри Мур (Geoffrey Moore), Брюс Мовери (Bruce Mowery), Нат Майерс (Nate Myers), Эд Нихаус (Ed Niehaus), Констанс Петерсен (Constance Petersen), Кит Плис (Keith Pleas), Роберт Райнман (Robert Reimann), Джон Ривлин (John Rivlin), Говард Рейнгольд (Howard Rheingold), Хайди Ройцен (Heidi Roizen), Нил Рубенкинг (Neil Rubenking), Пол Сафо (Paul Saffo), Джош Зайден (Josh Seiden), Расс Зигельман (Russ Siegelman), Донна Слоут (Donna Slote), Линда Стоун (Linda Stone), Тони Уокер (Toni Walker), Кевин Уикс (Kevin Weeks), Кевин Уэлш (Kevin Welch), Дэн Уиллис (Dan Willis), Хизер Уинкл (Heather Winkle), Стефан Уайлдстром (Stephen Wildstrom), Терри Виноград (Terry Winograd), Джон Цикер (John Zicker) и Пьерлуиджи Заппакоста (Pierluigi Zappacosta).

Этот проект «сроком на год» занял двадцать месяцев, и моя семья проявила величайшее терпение. Я в большом долгу любви и благодарности перед моей супругой Сью Купер и моими прекрасными юными сыновьями Скоттом и Марти. Люблю вас всем сердцем.

Предисловие

Спасайся кто может – мир захватывают компьютеры. Великолепные, мощные, они справляются с еще более важными задачами посредством ужасных, старомодных интерфейсов. Все больше проникая во все сферы жизни человека, они будут злить, раздражать нас, а некоторых способны даже убить. В ответ мы тоже будем безумно желать уничтожить наши компьютеры, но не осмелимся, потому что мы уже так сильно, неразрывно привязаны к ним, к этим монстрам, обещающим нам столь многое и без которых современная жизнь уже немыслима.

К счастью, выход есть. Мы должны фундаментально переосмыслить взаимодействие компьютера и человека. Изменить эти взаимоотношения на более глубоком уровне, применяя новаторские подходы, потому что причина наших все возрастающих проблем кроется вовсе не в машинах, а внутри нас самих. Именно люди создали интерфейсы, которые нас так раздражают; люди продолжают использовать ставшие нефункциональными машины, невзирая на то, что эти ужасные интерфейсы утомляют глаза, приводят к боли в спине и вредят сухожилиям в запястьях. Как следует из названия этой книги, все мы стали безумцами, обитающими в технологическом сумасшедшем доме, и создали этот безумный мир мы сами.

Эта книга – наш путь к спасению. А точнее, целью Алана Купера было показать, что дверь на свободу и без того широко раскрыта. Мы вольны выйти в любой момент, но, ослепленные безумием, мы до сих пор не замечали выхода. Весь секрет в том, чтобы переопределить способ, которым мы взаимодействуем с компьютерами, и сделать это в более широком контексте.

Сам Алан Купер – не просто такой же безумец, как и все мы; он еще и еретик, чьи взгляды способны разъярить тех, кто намерен держать нас в заключении. А желают этого инженеры, которые и разрабатывают столь ненавистные нам системы и по-прежнему придерживаются мнения, что единственный выход из этого ужаса – еще больше совершенствовать интерфейсы. Только сама концепция *интерфейса* – это реликвия из той поры, когда компьютеры были не столь распространены, их производительность была слабой, а сами они не слишком эффективно взаимодействовали с людьми, которые ими управляли. Интерфейсы были полезны, когда все взаимодействие с пользователем происходило через неизведанную территорию на стеклянном экране компьютера. Теперь же, в эпоху, когда компьютеры проникают в каждый уголок нашей жизни, мыслить категориями интерфейсов становится даже опасным. Компьютеры больше не используют интерфейсы

лишь в качестве «прослойки» между пользователем и их внутренним устройством – теперь они взаимодействуют с людьми, и такое взаимодействие будет лишь переходить на более глубокий уровень, становиться все более незаметным и более критичным для здравомыслия нашего общества и всеобщего выживания.

Алану Куперу разница между интерфейсом и взаимодействием знакома как никому другому. Его идеи произрастают из многолетнего опыта по содействию в проектировании продуктов, мягко и ненавязчиво заполоняющих мир вокруг нас. Он апробировал свои идеи на практике в течение долгих лет, и теперь, наконец, у него появилось время преобразовать полученный практический опыт в четкое описание того вызова, с которым мы столкнулись, в методологию побега из сумасшедшего дома, который мы воздвигали с такой любовью. Читайте эту книгу, и вы станете свободными.

Пол Саффо (Paul Saffo), директор Института будущего

Предисловие к оригинальному изданию

Книга-обоснование

Изначально я собирался написать совершенно другую книгу – книгу-руководство о процессе проектирования взаимодействия. В мае 1997 года, во время поездки в Тоскану двое моих друзей, Дон Маккини (Don McKinney) и Дэйв Карлик (Dave Carlick), уговорили меня написать книгу, которую вы держите в руках. Они убедили меня, что следует, прежде всего, обращаться к деловой аудитории.

Они знали о моем намерении написать руководство, но сомневались в востребованности книги о проектировании взаимодействия (interaction design). Они хотели, чтобы я написал книгу о том, насколько важен и нужен этот процесс. Предложение друзей было, конечно, интересным, но я не был уверен, что смогу.

Однажды поздней ночью на веранде нашей общей виллы с видом на Флоренцию у меня состоялся серьезный разговор с Дэйвом и Доном. На столе несколько пустых бутылок из-под кьянти и остатки хлеба, сыра и оливок. Сияют звезды, светлячки танцуют на лужайке, а вдалеке подмигивают огни древних куполов столицы Тосканы. Дэйв снова предложил мне отказаться от идеи книги-руководства и вместо этого «дать деловое обоснование проектированию взаимодействия».

Я энергично запротестовал: «Дэйв, я же не знаю, как писать такую книгу». И начал загибать пальцы: «Мне придется объяснять, насколько мерзок тот процесс разработки, что существует сейчас, как компании теряют деньги на неэффективном создании программного обеспечения, как ненадежны неудовлетворенные клиенты и как все эти проблемы способен разрешить более совершенный процесс проектирования».

Дэйв перебил меня: «Алан, это называется главами».

Его ответ лишил меня сил сопротивляться дальше. Я вдруг осознал, что повторяю избитый сценарий и что Дэйв прав. Книга, содержащая «бизнес-обоснование», необходима и более своевременна, чем книга-руководство. Дэйв и Дон убедили меня, что я действительно способен написать такую книгу.

Инженер, разбирающийся в бизнесе, либо бизнесмен, разбирающийся в технологиях

Успешный профессионал XXI века – это либо инженер, разбирающийся в бизнесе, либо бизнесмен, разбирающийся в технологиях, и именно такому человеку адресована моя книга.

Бизнесмен, разбирающийся в технологиях, понимает, что его успех зависит от качества доступной ему информации и его умения воспользоваться этой информацией. Инженер, разбирающийся в бизнесе, – это предприимчивый конструктор или ученый, специализирующийся на технологиях, но при этом обладающий деловой хваткой и осознающий, какая огромная сила заключена в информации. Оба новых архетипа будут доминировать в современном бизнесе.

Всех деловых людей можно разделить на две категории: на тех, кто овладеет высокими технологиями, и тех, кто скоро выйдет из бизнеса. Руководители больше не могут делегировать обработку информации специалистам, ведь бизнес и есть обработка информации. Сегодня вы можете выделиться качеством своих систем обработки информации, но не качеством систем производства. Если вы производите что-либо, скорее всего, в вашем продукте есть микросхема. Если вы предлагаете услугу, то, скорее всего, вы используете компьютеризованные инструменты. Попытка определить, какие предприятия зависят от высоких технологий, столь же глупая затея, как попытка определить, какие предприятия зависят от телефона. Революция высоких технологий затронула все сферы деловой деятельности, а цифровая информация стала основой ваших трудовых будней.

Кто-то сказал: «Человеку свойственно ошибаться, но чтобы провалить дело капитально, нужен компьютер». Неэффективные механические системы способны приносить незначительные убытки на каждой производимой детали, однако из-за некачественных информационных процессов можно потерять целую компанию. Влияние на вашу компанию продуктов, основанных на программном обеспечении, как и влияние инженеров, создающих эти продукты, огромно.

К сожалению, наши цифровые средства сложны для изучения, понимания и применения; они часто препятствуют достижению наших целей. Мы теряем деньги, время, возможности. Будучи инженером, понимающем в бизнесе, или же бизнесменом, понимающем в технологиях, вы создаете или потребляете продукты, основанные на программном обеспечении, а вернее всего, потребляете и создаете одновременно. Обладание более совершенными, более простыми в освоении и применении высокотехнологичными продуктами – в ваших личных и профессиональных интересах. Более качественные продукты не требуют больших временных и денежных затрат на свое создание. Ирония в том, что они не должны быть сложными и являются таковыми лишь потому, что устарели процессы их создания. Лишь давние традиции, основанные на заблуждениях, мешают нам сегодня получить более качественный продукт. Эта книга покажет вам, как можно требовать и получать лучшие продукты – продукты, которых вы заслуживаете.

Идея этой книги незамысловата: мы можем создавать мощные и приятные продукты, основанные на программном обеспечении, используя простой прием: *проектируя* взаимодействие продукта с пользователем *до* этапа программирования. Сегодня мы этого не делаем. Проектирование интерактивных продуктов, использующих программное обеспечение, – специализация столь же сложная, как собственно разработка.

*** * ***

Сделав выбор в пользу книги-обоснования и отказавшись от книги-руководства, я прошу прощения у каждого проектировщика взаимодействия, который возьмет в руки эту книгу. Из уважения к деловой части аудитории она содержит лишь краткие отступления в область практической методологии проектирования взаимодействий (в основном в главах части IV). Я включил лишь необходимое количество информации, чтобы показать, что такая методология существует, что она применима в любой предметной области и что ее преимущества очевидны всем, независимо от технических знаний.

Алан Купер,
Пало-Альто, Калифорния
www.cooper.com
inmates@cooper.com

Предисловие ко второму изданию

Недавно я познакомился с топ-менеджером одной из крупнейших технологических компаний мира. Официальное название его должности – вице-президент по вопросам простоты использования, и он несет ответственность за очень многие программные продукты, как небольшие, так и крупные. Это выдающийся и состоявшийся человек, выходец из сообщества HCI (Human-Computer Interaction, взаимодействие человека и компьютера), где принят формализованный подход. Он, как и его компания в целом, искушен в «юзабилити» – в тестировании и наблюдении из-за односторонних зеркал. Однако он пришел поговорить со мной не о тестировании, а о проектировании, и не о пользователях, а о персонах. Он рассказал, что в его компании полностью прекращено тестирование юзабилити на завершающей стадии разработки; вместо этого усилия прикладываются до начала разработки – при проектировании. Более того, он упомянул, что все его люди, умеющие и привыкшие наблюдать за пользователями в искусственных условиях, проходят переподготовку и будут заниматься этнографическими исследованиями в «полевых» условиях.

Этот топ-менеджер и его компания – символ тех огромных изменений, которые произошли в отрасли за пять коротких лет с момента выхода в 1999 году первого издания книги *The Inmates Are Running the Asylum* («Психбольница в руках пациентов»). Книга стала одновременно и манифестом для революции, и учебником. Не сосчитать, сколько писем я получил от менеджеров среднего звена, в которых рассказывалось, как после прочтения книги они покупали по экземпляру каждому из своих руководителей. Между тем разработчики программного обеспечения и университеты восприняли три главы части IV как исходный материал для руководства «сделай сам» по претворению в жизнь целеориентированного (Goal-Directed®) проектирования с использованием персон.

Я глубоко признателен всем менеджерам, программистам, руководителям и специалистам по юзабилити за то, что они воспользовались моими идеями и вывели юзабилити из лаборатории в жизнь, сместили фокус с тестирования на проектирование. Благодаря их усилиям полностью изменилась профессия специалиста по юзабилити. Сегодня в большинстве организаций, с которыми я контактирую, на полной ставке работают один или несколько профессиональных проектировщиков взаимодействия, и влияние этих людей на качество и поведение программных продуктов и услуг постоянно растет. Мне приятно осознавать, что эта книга способствовала их успеху.

Помню, как читал программную речь на конференции программистов в 1999 году, вскоре после опубликования книги. Название речи совпадало с названием книги, и свою речь я начал со слов, что «психбольница находится в руках пациентов, и эти пациенты – вы». В аудитории воцарилась гробовая тишина: 2500 инженеров пытались осмыслить мое обвинение. Я продолжил представлять основные идеи книги, и час спустя это сборище *Хомо логикус* настолько прониклось моими аргументами, что мне аплодировали стоя. Удивительно, но большинство программистов с энтузиазмом восприняли идеи

проектирования и проектировщиков взаимодействия. Они понимают, что нуждаются в помощи там, где речь идет о человеческой стороне конструирования программ, и счастливы получить, наконец, некоторые полезные указания. Программисты признают, что любая практика, повышающая качество и успех программ, не является для них угрозой.

В прошлом руководители считали проектирование взаимодействия задачей программирования и делегировали ее решение программистам, которые прилежно трудились над ней, хотя их опыт, подготовка, образ мышления и рабочий график не позволяли добиться успеха. В духе диагностики проблем эта книга подробно описывает такой провал, который всегда оказывается провалом программиста. Некоторые из программистов восприняли книгу враждебно, подумав, будто я злословлю или пытаюсь переложить на программистов вину за некачественный софт. Определенно, некачественные программы создаются при их *участии*, но они никоим образом не заслуживают осуждения. Я не виню программистов за сложные в использовании программы, и мне очень жаль, что у некоторых из них сложилось обратное впечатление. За некоторыми исключениями, знакомые мне программисты прилежны и добросовестны в своем желании угодить конечным пользователям, и стремятся неустанно повышать качество программ. Подобно пользователям, программисты – лишь жертвы несовершенного процесса, характеризующегося цейтнотом, противоречивыми указаниями и недостаточно эффективным руководством. Мне очень жаль, если у кого-то из программистов сложилось впечатление, будто я их обвиняю.

Жесткий процесс создания программ, в особенности высокая стоимость программирования и низкое качество взаимодействия, – проблема, попросту говоря, не технического плана. Это результат применения бизнес-практик в том направлении, в котором они устарели, – в разработке программ. С чистым сердцем, благими намерениями и с благословения руководства программисты пытаются решить эту проблему инженерным путем. Но работать интенсивнее здесь не помогает. Программисты ощущают все большую тщетность своих усилий, и их отчаяние нарастает.

Из нескольких своих недавних поездок я сделал вывод, что тревога в сообществе программистов нарастает. К сожалению, хуже всего чувствуют себя лучшие и опытные программисты. Они прикладывают титанические усилия, но излучают цинизм и впадают в тоску, понимая, что их умения растрачиваются попусту. Они могут и не знать точно, как именно получается, что их квалификация не находит правильного применения, но они не могут не видеть очевидного. Многие из лучших программистов вообще перестали программировать, поскольку работа раздражает их. Они ушли в преподавание, религию, писательство, консультационную сферу, потому что эти занятия не оставляют ощущения пустой траты времени и сил. Но этих трагических потерь вполне можно избежать. (В некотором смысле движение свободных программ с открытым кодом можно назвать раем для этих отчаявшихся программистов – тут они могут писать код по своим стандартам и быть судимыми только равными, не выслушивая советы от маркетологов или руководителей.)

Программистам не хватает времени, четких указаний и адекватных планов, позволяющих добиться успеха. Эти три кита – на совести руководителей, именно они не дают всего этого программистам, причем вовсе не по злому умыслу или по глупости, а по причинам, которых можно было бы избежать. Они просто не вооружены адекватными инструментами, позволяющими решать сложные и уникальные проблемы информационной эпохи. Ну вот, звучит так, будто я снова кого-то критикую, но на этот раз в мой прицел попали бизнесмены, а не программисты. Повторюсь, чтобы решить проблему, ее следует сначала разобрать на составляющие. Я ищу решения, а не козлов отпущения.

Мудрый руководитель Питер Друкер (Peter Drucker) в свои девяносто два года, большую часть которых он провел, направляя действия руководителей, смотрит на эту проблему со своей, уникальной точки зрения. В недавнем интервью журналу *CIO* он упомянул о наивном оптимизме руководителей в 1950-е – 1960-е годы, когда компьютеры только прокладывали дорогу в деловой мир. Эти руководители думали, что компьютеры

«окажут огромное воздействие на способы ведения бизнеса», но Друкер утверждает: «Но так не случилось. Очень немногие руководители задавали вопрос: „Какая информация мне требуется для выполнения данной работы?“» Компьютеры дали руководителям небывалые объемы данных, но лишь немногие поинтересовались, подходят ли эти данные для управления корпорацией. Образ существования бизнеса менялся очень быстро, однако менеджмент остался прежним. Друкер нападает на наши устаревшие бухгалтерские системы, рожденные в эпоху меркантилизма, повзрослевшие в век пара и стали и угасающие на пороге XXI века, эпохи информации. Друкер говорит: «Самая нужная вам информация – информация о внешнем мире, и этой информации у вас нет».

В последние несколько лет XX века, по мере раздувания мыльного пузыря доткомов, целые цистерны чернил уходили на продажу идеи, что в интернете существует «новая экономика». Знатоки утверждали, что продажа вещей во Всемирной паутине, где магазины строят из страниц, а не из кирпичей, принципиально отличается от привычных стилей бизнеса и что «старую экономику» уже не оживить. Разумеется, почти все компании новой экономики мертвы, финансисты, поддержавшие их, пережили шок, а эксперты, пропагандировавшие новую экономику, теперь заявляют, что то была пустая мечта. Новая-преновая линия такова, что нам суждено пока оставаться со старой-престарой экономикой.

Вообще говоря, я считаю, что мы живем при новой экономике. Более того, я думаю, что доткомы никогда не были ее частью. Напротив, они стали последним вздохом *старой* экономики, экономики производства.

В промышленную эпоху, до появления программ, продукты *создавались* из реальных материалов – из атомов. Затраты на добычу, плавку, приобретение, транспортировку, нагрев, формовку, сварку, окраску и снова транспортировку преобладали над всеми прочими расходами. В бухгалтерском учете эти расходы называются «переменными затратами», поскольку они различны для каждого продукта. «Фиксированные затраты», как вы, наверное, догадываетесь, очевидным образом не меняются и включают такие затраты, как корпоративное администрирование или начальная стоимость завода.

Классические правила управления бизнесом обусловлены производственными традициями промышленной эпохи. Однако эти правила не учитывают новые реалии эпохи информационной, в которой продукты уже не создаются из атомов, а состоят в основном из программного кода, из наборов битов. А биты не подчиняются тем же экономическим правилам, что атомы. Некоторые фундаментальные истины остаются справедливыми и для новой экономики. Цель любого бизнеса – стабильная прибыль, и есть лишь один законный способ получать ее: продавать товары или услуги дороже, чем обходится их создание. Из этого следует, что есть два пути повышения прибыльности: снижение затрат или рост прибыли. В старой экономике лучшим способом было снижение затрат. В новой экономике куда лучше работает рост прибыли.

Сегодня наиболее нужные и дорогие продукты состоят (полностью или почти полностью) из программного обеспечения. Они не требуют сырья. У них нет стоимости производства. Они не требуют затрат на транспортировку. Не требуют литья, обработки, покраски. Вот она, реальная разница между экономикой промышленной эпохи и экономикой эпохи информационной: в информационную эпоху практически или совсем отсутствуют переменные затраты, тогда как в позднюю промышленную эпоху именно эти затраты были главным фактором. Очевидно, что именно отсутствие переменных затрат делает новую экономику новой.

Зарплата программистов в вашем штате – фиксированные затраты или переменные? Один час работы программиста нельзя связать с одной продажей продукта – один и тот же код можно продавать много раз. Вложение в программирование можно амортизировать продажей миллионов копий продукта – точно так же, как продажа продуктов, созданных на заводе, амортизирует вложения в этот завод.

Стоимость создания программ не переменна, но и не фиксирована тоже. Разработка

программ – это непрерывный процесс для компании, приносящий прибыль, и это совсем не то же самое, что строительство завода. Высокооплачиваемые строители завода после завершения работ уходят на другую рабочую площадку. Программисты стоят гораздо дороже плотников и сварщиков и никогда не исчезают, потому что, по всей видимости, их работа никогда не кончается. Кто-то может сказать, что программирование – это научно-исследовательские работы, и сходство действительно есть. Однако же научно-исследовательская работа – это гипотезы и эксперименты, призванные оценить теоретическую жизнеспособность продукта, и они происходят совсем не так, как настоящее создание продуктов. Эта мысль подтверждается тем, что традиционный бухгалтерский учет разделяет исследования/разработку и ежедневную деятельность, приносящую прибыль. Создание программ не попадает ни в одну из этих категорий учета, приемлемых для прежних предприятий.

Да, этим маленьким терминологическим несоответствием можно было бы и пренебречь как придиркой, уместной в беседе счетоводов за кружкой пива, но в действительности оно оказывает огромное влияние на финансирование разработки программ, на управление проектами по разработке и, что самое важное, на то, как к разработке программ относятся топ-менеджеры.

Программисты создают приложения, а руководители создают потоки прибылей и структурные подразделения. Программисты оценивают свой успех по качеству продукта, а руководители – по прибыльности вложений. Эту прибыльность они оценивают на языке математических терминов, позволяющем учитывать фиксированные затраты, переменные затраты, затраты на корпоративное администрирование, исследования и разработку, но, к сожалению, не описывающем подходящие модели для программ и программирования. Бухучет – основной язык бизнеса, и перечисленные категории настолько фундаментальны для всех оценок предпринимательской деятельности и коммуникации в бизнесе, что современные руководители полностью их усвоили. Программирование для них – еще одна статья корпоративных расходов, которую следует причислить к одной из существующих категорий. На практике большинство руководителей расценивают программирование как производственный процесс, имеющий переменные затраты. (Для целей налогообложения в большинстве компаний-разработчиков ПО программирование проходит по статье исследований и разработок, но во всех остальных отношениях расценивается как деятельность с переменными затратами.) Это худший выбор из всех возможных, поскольку он наносит серьезный ущерб процессу принятия бизнес-решений.

В промышленную эпоху основным преимуществом была массовость, которая позволяла снижать цены и делать продукцию доступной для широких слоев населения. Покупатель при этом получал функции, которые ранее были не доступны или доступны для богатых людей, поскольку производились вручную. Компании конкурировали в области продажных цен, непосредственно связанных с переменными затратами – затратами на производство и доставку. В информационную эпоху доступность продукции по разумным ценам считается обстоятельством само собой разумеющимся. В конце концов, программы можно распространять через интернет – практически бесплатно и почти не прилагая усилий.

Как вы помните, бизнес может повышать доходность за счет роста прибыли или сокращения расходов. Другими словами, предприятие может увеличивать инвестиции в области фиксированных затрат, повышая качество продукции и укрепляя таким образом ценовые позиции, или же снижать переменные затраты, что означает снижение стоимости производства. В старой, «атомной» экономике снижение затрат давалось легко и было эффективным и предпочтительным. Сегодня же руководители, ставящие программирование на одну доску с производством, воображают, будто снижение стоимости программирования дается так же легко и оказывается таким же эффективным. К сожалению, старые правила больше не действуют.

Поскольку производство программ сопровождается незначительными переменными затратами, снижение этих затрат не дает преимущества в бизнесе. С точки зрения бухгалтера

зарплаты программистов – переменные затраты, однако в действительности их зарплаты представляют собой долгосрочные вложения, фиксированные затраты. Снижение стоимости программирования и снижение стоимости производства – разные вещи. Первое можно сравнить, скорее, с раздачей работникам дешевых инструментов, чем со снижением зарплат. Компании, заказывающие разработку в других странах с целью снижения зарплат, просто не понимают сути дела.

Более того, единственно возможный вариант для экономического подъема – это повышение качества и, как следствие, привлекательности продукта или услуги, а повышения качества невозможно добиться, сокращая затраты на проектирование и программирование. Правда в том, что в исследования, анализ, планирование и проектирование следует вкладывать больше времени и денег, чтобы полученный результат лучше соответствовал потребностям потребителя.

Разумеется, такой подход требует мышления, непривычного для предпринимателей XXI века. Им следует не *снижать* затраты на создание *каждого* объекта в отдельности, а *повышать* затраты на создание всей *совокупности* объектов. В этом суть новой экономики, и именно об этом говорит Питер Друкер.

Современные фармацевтические компании, разрабатывающие высокотехнологичные лекарства, имеют кое-что общее с новой экономикой программного обеспечения. Действительная стоимость производства одной таблетки минимальна, однако разработка лекарства может обойтись в миллиарды долларов и продолжаться более десяти лет. Подъем после выхода на рынок нового волшебного препарата может длиться до бесконечности, а вот выпуск недоработанного препарата способен принести только катастрофический спад. Фармацевтические компании знают, что снижение издержек на разработку – нежизнеспособная стратегия.

Как и разработка лекарственных препаратов, разработка ПО совсем не похожа на строительство завода. Завод – физический актив, который принадлежит компании, а работников завода, как правило, легко заменить другими. Неосязаемые, но невероятно сложные паттерны мыслей, составляющие программное обеспечение, обладают ценностью только вместе с написавшим код программистом. Ни одна компания не может себе позволить относиться к программистам так же, как к заводу. Программисты требуют постоянного внимания и поддержки, причем гораздо большей, чем любой завод.

Чаще всего пытаются сэкономить на архитектуре программного продукта, а эта часть проектирования (во время которого изучаются пользователи, определяются сценарии работы, проектируется взаимодействие, определяется форма, описывается поведение) выполняется человеком. Конечно, иногда проектированию уделяют слишком большое внимание, но сокращение этой фазы пользы точно не приносит. Каждый доллар и каждый час, потраченные на архитектуру, принесут десятикратную экономию на этапе программирования. Кроме того, вложения в достаточно качественное проектирование сделают ваш продукт привлекательным, а это означает, что продукт принесет больше прибыли. Его привлекательность станет основой для вашего бренда, создаст возможности для повышения цен, сделает клиентов лояльными, подарит вашему продукту более долгую и плодотворную жизнь. И хотя здесь нет экономии средств, вы получаете большое преимущество в смысле качества. По иронии судьбы лучший способ увеличить прибыльность в информационную эпоху состоит в том, чтобы больше потратить.

К сожалению, в руководителях живет практически неистребимое желание расходовать на программирование поменьше времени и средств. Они не считают тактику сокращения затрат устаревшей и не понимают, что сокращение инвестиций в программирование крайне негативно влияет на качество, привлекательность и прибыльность продукта в долгосрочной перспективе. Разумеется, простым повышением затрат не добиться улучшений. Зачастую ситуация даже ухудшается, если деньги вкладываются бездумно, без анализа и правильного руководства. Мой первый наставник, Дэн Хоакин (Dan Joaquin), любил повторять, что старую истину «получаешь то, за что платишь» нужно поменять на «не получаешь того, за

что не платишь». Действия без планирования всегда чреваты риском потратить слишком много. Фокус в том, чтобы потратить правильное количество денег, а для этого нужно разбираться в управлении созданием программного обеспечения. Еще для этого нужны процессы, обеспечивающие руководителей пониманием и информацией для принятия верных решений. Дать компаниям такие процессы – цель этой книги.

В буме доткомов участвовали компании с бизнес-моделями, полностью ориентированными на снижение переменных затрат. Хотя многие доткомы рекламировали преимущества онлайн-покупок, их сайты, тяжеловесные и неудобные, были плохой альтернативой обычной поездке в торговый центр. Основатели доткомов просто лучились от восторга (и пресса, кстати, тоже), потому что им удалось создать предприятия розничной торговли с невероятно низкими переменными затратами. Феерический провал этих предприятий, несомненно, доказывает, что информационной эпохой правят иные экономические правила, чем промышленной эпохой.

В старой экономике более низкие переменные затраты приводили к более широкому распространению товара и снижению розничных цен. Это двойное преимущество было выгодно покупателям, а покупатели – фундамент экономического успеха промышленной революции. В новой экономике успех бизнеса зависит от способности дать потребителю что-то новое и более качественное. Реальное качество каждого шага транзакции – от просмотра страниц до сравнения товаров – должно быть ощутимо более высоким для пользователя. Гораздо приятнее сделать покупку обычным путем, чем продираться через 11 экранов и в конечном итоге выяснить, что все равно придется звонить в компанию. Покупки в сети становятся совершенно ненужными и непривлекательными, если требуется набрать свое имя, свой адрес и ввести информацию о кредитной карте три или четыре раза, а затем обнаружить, что сайт не позволяет купить все необходимое и все равно придется ехать в обычный магазин, сделанный из атомов. Сегодня простое снижение цены на продукт уже не дает гарантии успеха.

Компания Pets.com, специализирующаяся на продаже корма для собак через интернет, не предлагала более качественный корм, как не предлагала и шопинг, более приятный, чем в обычном зоомагазине; она не предлагала новую информацию, новые возможности, новую уверенность. Она предлагала лишь дешевую доставку, складирование и торговлю (все это переменные затраты) на сайте *Pets.com*. Компания применила классическую тактику снижения затрат, характерную для промышленной эпохи, проигнорировав фундаментальные принципы новой экономики. Конечно, это было еще не первое дыхание новой экономики, но для старой это были последние судороги. Я совершенно убежден, что любой товар можно продавать через интернет успешно и прибыльно. Для этого всего лишь нужно, чтобы в онлайн-магазине было намного приятнее покупать, чем в конкурирующих розничных сетях, и цена здесь – всего лишь один из факторов. Есть лишь один способ добиться этого: архитектуру системы следует создавать с целью максимально удовлетворить конечного пользователя. Нельзя относиться к любому аспекту проектирования и создания программного обеспечения как к производственному процессу – это повлечет за собой провал. Проектирование и программирование – это неподходящие цели для традиционных методов сокращения затрат. Конечно, можно потратить на создание программ слишком много времени и денег, но опасность потратить меньше необходимого гораздо серьезнее.

Скорее всего, эта опасность вам знакома и удивления не вызывает, но большинству топ-менеджеров крупных компаний сама идея кажется неприемлемой. Эти руководители до сих пор работают по моделям бухучета, вошедшим в моду в эпоху паровых машин, тогда как все аспекты жизни их компаний – функционирование, принятие решений, коммуникации и финансы – полностью зависимы от программного обеспечения. Термины и понятия, которыми оперируют эти руководители, не учитывают уникальную природу бизнеса в эпоху, когда инструменты и продукты торговли являют собой неосязаемые переплетения битов вместо железнодорожных составов, груженных сталью.

Корпорации уже нанимают проектировщиков взаимодействия и начинают применять

целеориентированный подход, однако качество программных продуктов не слишком улучшилось. Более того, высокие затраты на программирование и жесткий процесс создания программ никуда не делись. Почему?

Перемены невозможны, пока топ-менеджеры компаний не осознают, что проблемы с программами – это не технические сложности, а важные бизнес-вопросы. Наши проблемы останутся неразрешенными до тех пор, пока мы не изменим *процесс* и *организацию*.

Компании живут не только по устаревшим финансовым моделям, но еще и по негодной организационной модели. Эта модель скопирована с научной, в ней создание программы смешивается с планированием и решением инженерных задач. Такова природа исследований. Прискорбно, что эта парадигма без предупреждения была перенесена в неизменном виде в мир бизнеса, где ей не место.

Корни всех современных производственных дисциплин уходят в доиндустриальные времена. Всех, кроме дисциплины программного обеспечения, которая возникла, когда индустриализация уже закончилась. Только программирование вышло сразу из университетской среды, где время исследований не ограничивалось, студенческой рабочей силы было хоть отбавляй, о прибылях вообще не говорили, а неработающая программа могла сойти за весьма удачный эксперимент. Неслучаен тот факт, что *Microsoft, IBM, Oracle* и другие ведущие компании-разработчики ПО расположены в «кампусах». Университетам не нужны прибыли, они не стараются успеть создать привлекательный и полезный продукт к определенному сроку.

Любой бизнес, не связанный с программным обеспечением, начинается с исследований и заканчивается распространением продуктов или предложением услуг. Компания тщательно планирует время между этими двумя событиями, осознавая, что преждевременный выпуск непродуманного продукта опасен как для банковского счета, так и для репутации. Руководители знают, что время, размышления и деньги, вложенные в планирование, обернутся крупными дивидендами – отлаженным и быстрым процессом производства, популярностью и прибыльностью конечных продуктов.

Во всех других конструкторских дисциплинах инженеры создают стратегию, а ремесленники претворяют ее в жизнь. Инженеры не строят мосты сами, этим занимаются монтажники. Только в области программного обеспечения перед инженером стоит задача создать собственно продукт. Только в области программного обеспечения перед «монтажником» стоит задача определить, как следует создавать продукт. Только в области программного обеспечения эти две задачи решаются не последовательно, а одновременно. Однако компании-разработчики ПО, похоже, не осознают существования такой аномалии. Инженерное дело и конструкторское дело так тесно пересекаются, что специалисты и руководители их не разделяют и, вероятно, не различают. Планированием любого рода здесь пренебрегают или откладывают его до тех пор, пока не станет слишком поздно. Считается нормальным откладывать решение очень сложных инженерных проблем до момента, когда экономически станет накладно откатываться к фазе проектирования, потому что полным ходом пишется код для коммерческой версии продукта.

Проектирование архитектуры следует начинать на ранних стадиях инженерного планирования. Более того, именно оно должно быть движущей силой на этих стадиях, но такие разработки обычно откладываются до момента старта проекта и ведутся параллельно с созданием кода, поэтому не занимают должного места в процессе конструирования продукта. Компании нанимают проектировщиков взаимодействия и обучают специалистов по юзабилити создавать персон, однако работа этих людей почти не влияет на стоимость разработки и качество завершенного продукта.

Решение проблемы – в руках президентов и генеральных директоров корпораций. Делегируя решение своим техническим директорам или вице-президентам по разработке, они поступают неверно. Эти достойные исполнители – технари, а проблема не техническая. Как сказал Друкер, инструменты для бухгалтерии, на которые полагаются директора компаний, попросту не отражают истинной природы этих компаний. Нельзя ведь на основе

точных показаний спидометра утверждать, что автомобиль движется в нужном направлении. В мире бизнеса цифровых технологий такой подход не может быть эффективным.

Одна из серьезных проблем применения неверных методов бухучета и организации для разработки программ состоит в том, что руководители не видят, сколько денег, израсходованных на программирование, потрачено впустую. Точная система показала бы, что из каждого доллара около пятидесяти центов тратится неправильно и что еще два или три доллара требуется, чтобы исправить проблему, вызванную этим некорректным вложением средств. В любом другом бизнесе подобная статистика вызвала бы революцию, однако отрасль программного обеспечения продолжает жить в состоянии блаженного неведения.

За последние тринадцать лет моя компания *Cooper* проконсультировала сотни компаний. Мои талантливые проектировщики создали для большинства клиентов «чертежи» продуктов, позволяющие коренным образом изменить ситуацию, но лишь немногие сумели воспользоваться всеми полученными преимуществами. В большинстве этих компаний проектирование взаимодействия и архитектуру программ считают лишь рекомендацией, в этих компаниях последнее слово всегда остается за программистами и инженерами. Ни один из президентов этих компаний не имеет ни малейшего представления о том, что происходит в офисах инженеров, и потому расписание ужимается безо всякой причины. Программисты постоянно работают в условиях дефицита ресурсов, и прежде всего – времени на хорошее программирование, а также времени, чтобы определить, где вообще требуется программирование. Они вынуждены защищаться, отвергая советы, и изворачиваться, общаясь со своими менеджерами.

На мой взгляд, существует два типа исполнителей: инженеры и боящиеся инженеров. Первые множат знакомые проблемы, поскольку их точка зрения безнадежно затуманена конфликтом интересов. Вторые множат проблемы, поскольку не умеют говорить на языке программистов. И я не имею в виду языки Java и C#. Я имею в виду, что у предпринимателей и программистов нет общих инструментов и общих целей. *Хомо сапиенс* делегирует человеческие проблемы *Хомо логикус*, не осознавая, что решение могло бы оказаться куда лучше в случае применения – на исполнительном уровне – уместных финансовых и организационных моделей.

У компаний есть прекрасная возможность сдвинуться с мертвой точки и сосредоточить усилия на удовлетворении потребностей клиентов, а не на программах, на персонах, а не на технологиях, на выгоде, а не на программистах. Я с нетерпением жду, когда появится просвещенный руководитель, который ухватится за эту возможность и навсегда изменит способ создания программного обеспечения, подав смелый и успешный пример.

Алан Купер,
Менло-Парк, Калифорния,
октябрь 2003 г.
www.cooper.com
inmates@cooper.com

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Часть I. Компьютерная неграмотность

1. Загадки информационной эры

Что будет, если скрестить компьютер и самолет?

Декабрь 1995 года. Самолет авиакомпании *American Airlines* совершает вылет по регулярному рейсу 965, следуя из Майами в колумбийский город Кали. Подлетая к взлетно-посадочной полосе, пилот «Боинга-757» должен был выбрать радиомаяк ROZO, чтобы определить ее координаты. Воспользовавшись бортовым навигатором, пилот ввел первую букву названия радиомаяка – R. Навигатор незамедлительно вывел список ближайших радиомаяков, названия которых начинались с R, – пилоту оставалось лишь выбрать нужный. Он выбрал самый первый, ориентируясь на указанные рядом с названием, казалось бы, верные координаты маяка. К несчастью, это оказался маяк ROMEO, который находился в 212 километрах к северо-востоку. Рейс же летел в южном направлении, а в момент выбора маяка вошел в долину, которая тянулась с севера на юг, так что отклоняться от курса было крайне рискованно. Ориентируясь на показания бортового навигатора, пилоты приступили к корректировке курса в направлении востока, а спустя несколько минут «Боинг-757» на высоте 3000 метров врезался в вершину гранитного утеса. Погибли 152 пассажира и 8 членов экипажа. Выжили только четверо пассажиров, все из них получили тяжелые травмы. Как обычно бывает в подобных ситуациях, причиной крушения самолета, объявленной Национальной комиссией по безопасности, был человеческий фактор. Бортовой навигатор, который использовали пилоты для ориентировки, отобразил верные данные, только не те, что нужны были для успешной посадки самолета в аэропорту Кали. Фактически человеческий фактор действительно имел место – ведь именно пилот выбрал маяк, совершив роковую ошибку, однако общая оценка ситуации позволяет утверждать, что вины пилота здесь не было.

На передней панели бортового навигационного компьютера самолета был показан радиомаяк, на который ориентировался рейс, и индикатор отклонения от курса. Если курс взят верно, стрелка будет указывать в центр шкалы, но правильность выбора самого маяка не проверяется. Внешний вид индикатора одинаков как в ситуации посадки, так и при возможном крушении. В данном случае навигатор отобразил данные о верно взятом курсе, но не учел, что неверно выбранный маяк может стать причиной катастрофы.

* * *

Полнота и точность передаваемых данных может быть абсолютной, но при этом повлечь трагический исход из-за их некорректности в текущей ситуации. При взаимодействии с компьютерами это происходит повсеместно, а ведь в современном мире все сложнее найти сферу, в которой бы не применялись компьютерные устройства. Начиная от пассажирских самолетов и заканчивая потребительскими товарами и услугами – везде используются компьютеры и везде их применение сопровождается характерным способом взаимодействия и поведения.

Существует такая шутка, широко известная в компьютерной среде: пилот небольшого самолета заблудился в облаках. Он идет на снижение, пока не оказывается рядом с офисным зданием, и кричит человеку в открытом окне: «Не подскажете, где я?» На что человек выдает ответ: «Вы в самолете, примерно в тридцати метрах над землей». Пилот тут же берет верный курс и спустя некоторое время благополучно приземляется в аэропорту. Пассажиры самолета удивленно спрашивают, как он понял, куда лететь. И пилот говорит: «Тот человек ответил мне совершенно точно и правдиво, но эта информация была абсолютно бесполезна, поэтому я сразу догадался, что этот человек – разработчик программного обеспечения из *Microsoft*, а я знаю, где расположено здание *Microsoft* по отношению к аэропорту».

В свете описанной выше трагедии рейса 965 эта шутка звучит зловеще, тем не менее профессионалы из цифрового мира не упускают возможности в очередной раз рассказать ее и посмеяться, потому что она подчеркивает главную особенность компьютеров: они могут выдавать нам факты, но не разъясняют их. Их информация отличается точностью, но только лишь ее нам недостаточно, чтобы достичь намеченных целей. В бортовой навигатор рейса 965 легко можно было заложить функцию сообщать пилотам о неверном выборе маяка. Один простой намек на то, что выбор маяка ROMEO «нетипичен» или «незнаком», мог бы предотвратить крушение. Только пассажиры и сам рейс, вероятно, не были тем, что занимало ум бортового компьютера. Он был занят исключительно собственными вычислениями.

Сложности во взаимодействии с компьютерами влияют на всех нас, временами приводя к фатальным последствиям. Но программные продукты сложны в применении не из-за сложности самих компьютеров, а потому, что в основу их разработки заложен неверный процесс. Данная книга призвана не только продемонстрировать следствия такого неверного процесса, но и прояснить причины его возникновения. Далее мы рассмотрим, как следовало бы изменить процесс, чтобы наше программное обеспечение обрело дружелюбный вид и мощный функционал. В этой главе основной акцент прежде всего делается на серьезность рассматриваемой проблемы.

Что будет, если скрестить компьютер и фотокамеру?

Вот одна из загадок эры информации: что будет, если скрестить компьютер и фотокамеру? Верный ответ: компьютер! Тридцать лет назад[1] моим первым фотоаппаратом был 35-миллиметровый Pentax H, в него вставлялась маленькая батарейка, от которой запитывался датчик освещенности (экспонометр). От меня требовалось только менять батарейку раз в два года, как в наручных часах.

Пятнадцать лет назад моей первой электронной фотокамерой был 35-миллиметровый Canon T70, в который вставлялось уже две пальчиковые батарейки, за счет чего достаточно простой электронный блок экспонометра приводился в действие, а также подавалось питание для автопрокрутки пленки. Сэкономить заряд батареек помогал простой выключатель на корпусе фотоаппарата.

Пять лет назад у меня появился Logitech – цифровой фотоаппарат первого поколения; он тоже имел подобный выключатель, но в дополнение обладал чуть более продвинутой электронной начинкой, напоминавшей зачатки компьютерного разума. Теперь, когда я

[1] Считая от 2006 года, когда была издана книга. – *Примеч. ред*.

забывал его отключить, он сам автоматически выключался спустя минуту бездействия. Довольно мило.

Год назад внутри моей Panasonic PalmCam – цифровой фотокамеры второго поколения – находилась куда более умная компьютерная микросхема. Ее ум был настолько велик, что простой рычажок «Вкл/Выкл» эволюционировал в переключатель Off/Rec/Play, который включал разные режимы работы камеры. Режим Rec предназначался для фотосъемки, а просмотр фотографий на небольшом экране осуществлялся в режиме Play.

Последней фотокамерой, которой я обзавелся, стал цифровой фотоаппарат третьего поколения Nikon CoolPix 900, и он еще более сообразительный. Его электронный мозг настолько умный, что представляет собой полноценный компьютер, который даже показывает песочные часы в стиле Windows во время загрузки. Его переключатель, словно рыба-мутант, отрастил дополнительные «головы», и теперь их стало *четыре*. Каждая из них отвечает за свою функцию: Off/ARec/MRec/Play. ARec – это режим автоматической записи, а MRec – режим записи вручную. Как по мне, разницы между ними нет. При этом у камеры нет режима On (включено), так что даже мои друзья не смогли понять, как ее включить, без моих объяснений.

Эта новая камера потребляет достаточно много энергии, так что создатели предусмотрительно наделили ее изощренной функцией контроля уровня заряда аккумулятора. Теперь все обычно происходит так: я перевожу злосчастный переключатель Off в режим MRec, мучительно жду приблизительно семь секунд, пока камера загрузится, и только потом могу направить ее на предмет съемки. Далее я выбираю подходящий ракурс и приближение камеры для получения наилучшего кадра. И в тот самый миг, когда мой палец почти нажал на спуск, камера вдруг решает, что одновременное увеличение кадра, питание вспышки и поддержание яркости экрана способны вконец исчерпать заряд батареи. Порываясь сохранить последние капли жизненных сил, камера временно отключает функцию съемки. Но мне об этом неизвестно, мой взгляд направлен в видоискатель. Я воодушевляюще машу руками, прошу сказать «Сы-ы-ыр!» и нажимаю на спуск. Компьютерный блок фотокамеры отслеживает нажатие кнопки, но не способен ничего предпринять. Программа управления питанием делает еще одну слабую попытку спасти ситуацию и в мгновение ока принимает ответственное решение: уменьшить нагрузку. В результате этого энергоемкий LCD-экран гаснет. Я недоуменно гляжу на камеру, пытаясь понять, почему не получилось сделать снимок, пожимаю плечами и опускаю руку, в которой держу камеру. Однако сэкономленная на отключении экрана энергия позволяет дополнительно запитать другие элементы камеры. У программы управления питанием открывается «второе дыхание», и она решает, что *прямо сейчас* энергии хватит, чтобы сделать снимок. Она посылает сигнал основной программе, которая терпеливо ждет момента, когда можно будет продолжить процесс фотосъемки и выполнить ранее посланную мной команду сделать снимок. В результате камера делает великолепно сфокусированное, безупречно экспонированное высококлассное цифровое фото моего колена.

На моем старом механическом фотоаппарате Pentax все настройки фокусировки, экспозиции и выдержки нужно было устанавливать вручную, тем не менее это вызывало куда как меньше мучений, чем процесс фотосъемки на современный, полностью компьютеризованный Nikon CoolPix 900, в котором все эти функции автоматические. Nikon по-прежнему позволяет делать снимки, но его *поведение* характерно для компьютера, а не для фотокамеры.

*** * ***

Лягушке, оказавшейся в кастрюле с холодной водой на плите, невдомек, что повышение температуры смертельно. Это происходит оттого, что нарастающий жар притупляет ее ощущения. Мои фотокамеры медленно эволюционировали от простого к сложному, становясь все более компьютеризированными, а я не осознавал этого, совсем как

та лягушка. И это характерно для всех нас, испытывающих на себе это медленное, убаюкивающее чувства посягательство компьютеров и присущего им поведения на нашу жизнь.

Что будет, если скрестить компьютер и будильник?

Компьютер! Недавно я поставил в своей спальне новый дорогой будильник со встроенным радио марки JVC FS-2000. Внутри устройства кроется довольно сложная компьютерная начинка, обещающая высокую точность. Кроме того, оно обладает цифровым звуком и еще множеством функций. Теперь я могу просыпаться в точно заданное время под мелодию с компакт-диска и, более того, будильник настолько умен и деликатен, что звук этой мелодии не взорвет барабанные перепонки в шесть утра, а будет нарастать постепенно. Эта редкая в будильниках опция действительно приходится весьма кстати, и только мысль о ней усмиряет мое желание вышвырнуть это дьявольское изобретение из окна.

Понять, на какой день недели установлен звонок будильника, чертовски сложно, так что периодически он не будит меня в понедельник и нагло врывается в мой мирный сон ранним субботним утром. Конечно, в этом устройстве имеется индикатор активности будильника, но это вовсе не значит, что он пригоден для использования. Встроенный алфавитно-цифровой LCD-дисплей запутанно отображает всяческие обозначения функций в часах. Например, маленький значок часов в левом верхнем углу LCD-дисплея сообщает о том, что будильник активен. Только в полутемной спальне опознать этот значок весьма затруднительно. У дисплея имеется функция подсветки, благодаря которой значок часов становится заметным, но включается она лишь в том случае, когда работает радио или играет компакт-диск. При этом есть один нюанс: будильник, даже если его установить, не прозвенит при включенном проигрывателе компакт-дисков. Именно такое противоречивое поведение часто ставит меня в тупик.

Отключить будильник проще простого: для этого нужно только один раз нажать кнопку Alarm, и значок часов пропадет с дисплея. А вот чтобы включить будильник, потребуется нажать эту кнопку не меньше пяти раз. Когда нажимаешь ее первый раз, на дисплее появляется время, в которое должен прозвенеть будильник. На второй раз отображается время, когда звук будильника должен выключиться. На третьем нажатии нужно выбрать, зазвучит ли радио или же компакт-диск. А на четвертом – определить желаемый уровень громкости. Завершающее пятое нажатие закрывает меню установки будильника и приводит часы к обычному режиму. Здесь главное – не нажать еще один, лишний раз, ведь тогда будильник вовсе отключится. Однако полусонному человеку в темноте спальни безошибочно исполнить этот маленький цифровой кордебалет бывает весьма затруднительно.

В силу своей упрямой симпатии к подобного рода гаджетам я не оставляю попыток разобраться в этом устройстве, надеясь однажды победить. Чего не скажешь о моей супруге – она уже давно перестала пытаться найти подход к этому дьявольскому агрегату. Она восхищается сглаженными линиями в его современном дизайне и качеством звучания, но экзамен в качестве будильника он не прошел, потому что требуется слишком много усилий, чтобы заставить устройство делать то, что нужно. Эти часы хотя и могут разбудить меня, но *ведут себя* как компьютер.

В отличие от него, мой древний 11-долларовый некомпьютеризированный будильник разливался внезапным жутким дребезжанием, когда требовалось меня разбудить. О том, что он включен, можно было узнать по яркой красной лампочке. А когда он выключался, лампочка гасла. Тот старый будильник не нравился мне по ряду причин, но по меньшей мере мне было ясно, собирается он меня будить или нет.

*** * ***

По той причине, что производителям намного выгоднее применять для контроля внутренних процессов устройств компьютеры, нежели механизмы старого образца, компьютеризация устройств и процессов оказания услуг в нашей жизни экономически неотвратима. Это значит, что все существующие устройства скоро начнут вести себя как самые несносные компьютеры, если только мы не попробуем подойти к этому иначе.

*** * ***

Продуктами для конечного потребителя описанное выше явление не ограничивается. Устройства, снабженные компьютерной начинкой, и компьютеризированные сервисы отличаются от своих простых собратьев с ручным управлением широтой вариантов применения и количеством опций. Но в реальности люди чаще склонны использовать именно механические устройства и делать это более гибко, искусно и осознанно, нежели при обращении с их современными версиями, работающими на кремниевых микросхемах.

Высокотехнологичные компании, в стремлении улучшить свои разработки, сводят этот процесс к наполнению их сложным и часто ненужным функционалом. Но такой некорректный процесс не помогает повысить качество продуктов, а только добавляет новые функции, так что это именно то, чем в действительности и занимаются его создатели. Далее в этой книге я продемонстрирую, как совершенствование процесса разработки приносит больше счастья пользователям, без лишних затрат на никому не нужные дополнительные функции.

Что будет, если скрестить компьютер и автомобиль?

Компьютер! Роскошный высокотехнологичный спортивный автомобиль Boxster от *Porsche* оснащен семью компьютерами, посредством которых управляются его сложные подсистемы. Один из них целиком предназначен для управления двигателем. В память этого компьютера заложены специальные программы, которые помогают преодолевать критические ситуации. К несчастью, эти программы сами периодически становятся причиной странных сбоев в работе автомобиля. Некоторые ранние модели отличались такой особенностью: когда уровень топлива в баке доходил до предельно низкой отметки – где-то около четырех литров, – бензин скапливался у стенки бака под действием центробежной силы при крутых поворотах. Это приводило к попаданию воздуха в топливную систему. Компьютер определял, что в топливной смеси произошли серьезные изменения, и интерпретировал это как фатальный сбой системы топливного впрыска. Для предотвращения критических последствий компьютер отключал зажигание, и автомобиль останавливался. Более того, во избежание тех же последствий компьютер не позволял заново запустить двигатель до тех пор, пока не будет произведена буксировка машины в автомастерскую для устранения неисправностей.

Все, что могла посоветовать тогда компания *Porsche* владельцам ранних Boxster, столкнувшихся с этой проблемой, – это открыть капот и отсоединить аккумулятор минимум на пять минут, тогда компьютер удалил бы из памяти сведения об инциденте. Несмотря на то что спортивные автомобили позволяют разъезжать по двухполосным асфальтированным дорогам на предельной скорости, в крутых поворотах они начинают *вести* себя в точности

как компьютер.

*** * ***

В похвальном стремлении уберечь владельцев Boxster от всяческих бед разработчики программ для автомобиля только унизили их и заставили страдать. Каждый, кто любит гоночные автомобили, знает, что клиентская политика компании *Porsche* направлена на выражение уважения своим клиентам и предоставление большого количества привилегий. А случаи, подобные описанному выше, показывают, что программное обеспечение для автомобиля создавалось не той же самой *Porsche*, что выпускает другие его компоненты. Его создала компания внутри компании: программисты, а не легендарные немецкие инженеры автомобилестроения. По какой-то причине внедрение новых технологий привело к тому, что солидная компания с долгой историей пренебрегла своими ключевыми ценностями. Стандарты качества для проектировщиков программного обеспечения намного ниже, чем для более традиционных инженерных дисциплин.

Что будет, если скрестить компьютер и банк?

Компьютер! В те моменты, когда я снимаю денежные средства в банкоматах, я каждый раз наблюдаю одно и то же угнетающе запутанное поведение, столь характерное для компьютеров. Малейшая допущенная мной ошибка приводит к отмене всей транзакции и выкидывает меня из сеанса. Мне приходится вытаскивать карту, снова ее вставлять, снова набирать PIN-код и снова вводить запрос. И ошибку-то обычно я допускаю не по своей воле, а потому, что компьютер банкомата мастерски вводит меня в заблуждение.

Каждый раз он задает мне один и тот же вопрос: каким счетом я хотел бы воспользоваться для снятия наличных – текущим, сберегательным или депозитным, – и это учитывая то, что у меня всего один счет – текущий. Каждый раз я забываю правильный ответ, так что такой вопрос сбивает меня с толку. Приблизительно раз в месяц я нечаянно выбираю сберегательный счет, и адский агрегат без долгих размышлений прерывает сеанс, так что мне приходится начинать сначала. При этом для отказа в снятии наличных со сберегательного счета банкомат должен знать, что такой счет у меня отсутствует, однако он все равно предлагает мне его как один из вариантов. Единственное, что отличает меня в момент выбора банковского счета от пилота рейса под номером 965, который ошибочно выбрал маяк ROMEO, – это масштабы последствий.

Помимо этого, банкомат ограничивает мои действия, налагая суточный лимит снятия наличных, который составляет 200 долларов. Так что, даже если я успешно преодолею весь путь – аутентификацию, выбор типа счета, запрос нужной суммы – и это будет, к примеру, сумма, равная 220 долларам, компьютер банкомата, не церемонясь, отклонит транзакцию, выдав грубое сообщение о превышении суточного лимита. Он не скажет мне, какова величина этого лимита, не отобразит остаток на моем счете и даже не позволит ввести другую сумму меньшего порядка. Вместо этого он просто выплюнет мою карту и предоставит мне возможность начать все с «чистого листа», не снабдив меня ни каплей дополнительной информации, заставляя растущую очередь за моей спиной беспокоиться и раздраженно вздыхать. Реакции банкомата точны и основаны на фактах, но толку от них никакого.

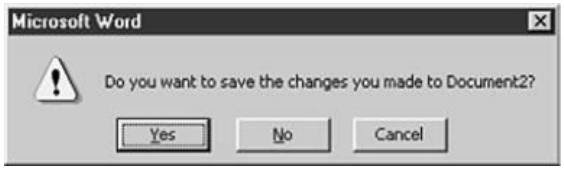
В память банкомата заложены определенные правила, которым он должен следовать. Я бы тоже рад следовать им, но это так по-компьютерному неблагоразумно – не сообщать мне об этих правилах, снабжать меня противоречивыми указаниями, а затем бесцеремонно наказывать меня за нарушение этих правил по чистому незнанию. Такое столь характерное для компьютеров поведение вовсе не является их истинной натурой.

Если быть точнее, никакой натуры у них нет: они просто выполняют операции по заданной программе. А программы такие же гибкие, как человеческая речь. Человек может

говорить грубыми или вежливыми словами, выражать поддержку или высказывать недовольство. Компьютер может так же легко, как и человек, сообщать сведения с должным уважением и учтивостью. Его только требуется этому научить. Увы, программисты не те люди, которые способны успешно обучить компьютеры таким вещам.

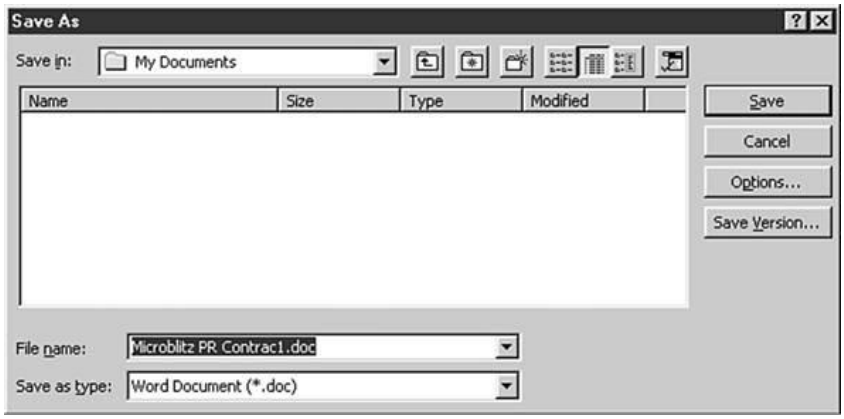
Как легко попасть в беду с помощью компьютера

Компьютер, воцарившийся на вашем рабочем столе, обладает именно таким несносным раздражающим поведением, какое и полагается компьютеру, для этого его даже не нужно ни с чем скрещивать. Моя подруга Джейн когда-то занимала должность координатора по связям с общественностью. На ее компьютере была установлена операционная система Windows 95 и программа Microsoft Word, в которой Джейн редактировала заметки и договоры. Основой Windows 95 являлась иерархическая файловая система. Все документы Джейн помещались в маленькие папки, которые, в свою очередь, помещались в другие маленькие папки. Джейн не понимала этого, так же как и не видела необходимости хранить файлы именно таким образом. Если быть точнее, Джейн вообще не слишком задумывалась об этом, а просто выбирала путь наименьшего сопротивления.



Получилось так, что Джейн только что закончила набрасывать черновик нового договора на пиар-услуги для стартапа из Кремниевой долины. Она выбрала опцию Закрыть в меню Файл. Вместо того чтобы просто выполнить команду и закрыть документ, программа Word вывела диалоговое окно. Как вы догадались, речь идет о всем знакомом вопросе Сохранить изменения в документе…? (Do you want to save changes…?). Джейн, как и обычно, нажала кнопку Да (Yes). Она слишком часто видела это окно, так что нажимала кнопки почти не глядя.

За первым диалоговым окном немедленно последовало следующее – и снова слишком хорошо знакомое Сохранить как… (Save As). В этом окне было великое множество непонятных для Джейн кнопок, значков и текстовых полей. Все, что она здесь могла понять и чем активно пользовалась, – это поле для ввода имени файла. Набрав подходящее имя, Джейн нажала кнопку Сохранить. Файл с договором сохранился в папке **Мои документы.** Джейн столько раз проделывала эту нудную процедуру, что уже даже не задумывалась о своих действиях.



В обеденное время, когда Джейн не было в офисе, Сунил, системный администратор компании, установил на ее компьютер обновленную версию антивируса VirusKiller 2.1.

Находясь за рабочим компьютером Джейн, Сунил открыл программу Word, чтобы просмотреть файл Readme для антивируса. Окончив просмотр файла, Сунил закрыл его и вернул компьютер Джейн в то состояние, в каком тот находился до обеда. По крайней мере, так ему казалось.

После обеда Джейн потребовалось снова открыть файл договора для распечатки и демонстрации руководителю. Джейн выбрала опцию Открыть в меню Файл, после чего на экране возникло диалоговое окно Открыть. Согласно ожиданиям Джейн, все ее договоры и другие документы должны были удобно отобразиться в этом окне в алфавитном порядке. Однако вместо этого ее взору явилось множество файлов, имен которых она никогда до этого не видела и не смогла опознать. Название одного из этих файлов значилось как **Readme.doc**.

Конечно, все дело было в том, что, когда Сунил открыл Word для просмотра файла Readme, он поручил программе отыскать неведомую папку на шестом уровне вложенности файловой системы и нечаянно изменил привычное для Джейн местоположение **Мои документы** на другое.

Ситуация на экране поставила Джейн в тупик. Первое, о чем она нерадостно подумала, что вся большая работа, которую она проделала, просто не сохранилась. Джейн взволнованно обратилась за помощью к Рене – своей подруге и коллеге, но Рене пришла в не меньшее замешательство. В итоге, дойдя почти до полного отчаяния, Джейн набрала номер Сунила, чтобы попросить помощи у него. Но Сунила на месте не было, а поймать его и вернуть все на круги своя удалось только в понедельник утром. В результате Джейн, Рене, Сунил и PR-компания в целом потеряли каждый по половине рабочего дня.

И хотя такие иерархические файловые системы нужны операционным системам компьютеров, людям они без надобности. Нет ничего удивительного в том, что больше всего древовидная структура каталогов нравится программистам, как нет и ничего примечательного в том, что обычным пользователям, таким как Джейн, это не приносит никакого удовольствия. А точнее, это непримечательно для всех, кроме тех самых программистов, которые разрабатывают подобное программное обеспечение. Они закладывают в программы такое поведение и отображение информации, что оно понятно для них самих, но между ним и тем, что подходит для Джейн, простирается большая пропасть. Вина за несоблюдение сроков и низкую эффективность труда ложится на Джейн, а не на разработчиков программного обеспечения, чей продукт привел к провалу проекта.

Как бы то ни было, Джейн работой обеспечена. Но ведь ко многим другим людям отношение как к недостаточно «компьютерно грамотным», так что их отказываются принимать на работу. Чем для большего количества должностей требуются навыки взаимодействия с компьютерами, тем шире становится разрыв между людьми, потенциально подходящими для этих позиций, и теми, кто не подходит, и тем сложнее его преодолеть. Политики могут сколько угодно требовать создания рабочих мест для малоимущих, только компании не дадут ни единого шанса сесть за их компьютеры людям, не имеющим достаточного опыта в этом деле. Их нужно слишком долго обучать, и остается большой риск порчи информации и причинения вреда бесценным базам данных.

Несносное поведение и невнятность взаимодействий, которыми отличаются продукты, работающие на программном обеспечении, уже настолько вошли в норму, что я называю происходящее «софтверным апартеидом». Из-за такого положения совершенно нормальные в целом люди не могут найти работу и устроиться в обществе, так как не владеют компьютером на приемлемом уровне.

Социальные активисты нашего просвещенного общества трудятся не покладая рук, пытаясь преодолеть расовые и классовые барьеры, пока специалисты индустрии технологий не менее тяжким трудом воздвигают еще более высокие заслоны. Но целенаправленная деятельность по проектированию таких программных продуктов, которые бы обладали большей человечностью и меньшей требовательностью, автоматически сделает их более близкими к народу, вне зависимости от класса и цвета кожи пользователей.

Коммерческому программному обеспечению тоже нелегко

Компьютеры распространяют свое влияние не только на кабины пилотов в самолетах, но и на пассажирские салоны, показывая там свой упрямый и своевольный характер, так хорошо знакомый нам. В современных авиалайнерах устанавливают системы развлечений в полете, через которые пассажиры могут слушать музыку и смотреть фильмы. Такие системы представляют собой обычные компьютеры, соединенные в локальную сеть, как те, что стоят в вашем офисе. Системы развлечений с расширенным набором опций обычно устанавливают только на больших самолетах, следующих трансокеанскими рейсами.

Бортпроводники одной из авиакомпаний были так расстроены использованием встроенной системы развлечений, что многие из них настоятельно требовали перевести их на более короткие местные рейсы, лишь бы не изучать ее сложный функционал. И это в высшей степени примечательно, потому что карьерный рост на авиалиниях – это всеми почитаемый процесс, происходящий в порядке старшинства, и именно эти маршруты дальнего следования всегда расценивались как самые лакомые кусочки: это было вызвано продолжительными остановками в экзотических уголках планеты, таких как Сингапур и Париж. Поэтому такое стремление стюардов и стюардесс мотаться туда-обратно на ничем не примечательных и совсем не романтических рейсах из Денвера в Даллас или Лос-Анджелеса в Сан-Франциско лишь для того, чтобы избежать взаимодействия с системой развлечения в полете, говорило о полном упадке боевого духа. Любая авиакомпания, которая когда-либо вынуждала своих самых ценных сотрудников – тех, что проводят с клиентами больше всего времени, – страдать из-за плохого оборудования, делала самую большую ошибку в своей жизни, попусту растрачивая денежные средства и рискуя доверием лояльных потребителей и собственного персонала.

Другая крупная авиакомпания установила еще более несносную компьютерную систему развлечений. К просмотру фильмов через систему этой компании была привязана функция оплаты. Ранее процесс сбора денег протекал в довольно расслабленном режиме: трудно было покинуть замкнутое пространство реактивного самолета на высоте одиннадцати тысяч метров, не оплатив услуги. Бортпроводники выполняли запросы пассажиров в товарах и услугах, а оплату собирали позже, когда их руки не были заняты подносами и когда пассажиры прекращали от них чего-то требовать. Такая процедура уберегала стюардов от беспрестанных хождений туда-сюда по узким проходам салона. Разумеется, случайные ошибки периодически возникали, но это никогда не превышало нескольких долларов. Имеющаяся система была довольно дружелюбной и терпимой; все были счастливы и довольны работой.

Новая компьютеризированная система оплаты услуг доставки медиаконтента привела к тому, что бортпроводникам сначала требовалось получить деньги от пассажиров, потом пройти через весь салон к его началу, где установлена панель управления для стюардов, затем ввести пароль обслуживающего персонала, а затем провести оплату. Только по завершении транзакции пассажир мог начать просмотр фильма или прослушивание музыки. Такой бессмысленный подход к проектированию системы привел к тому, что бортпроводники были вынуждены сотни лишних раз пройти по узким проходам за один только рейс. Вконец отчаявшиеся бортпроводники придумали устраивать короткое замыкание перед каждым рейсом долгого следования ровно через несколько минут после взлета. Далее следовало вежливое объявление с извинениями за сбой в работе системы и сообщением о том, что посмотреть фильмы на *этом* рейсе не получится.

Миллионы долларов затратила авиакомпания на то, чтобы разработать настолько бестолковую систему, что пользователи были готовы умышленно отключить ее, избегая всяческих взаимодействий с ней. Невинными жертвами этого процесса стали тысячи пассажиров, скучающих во время полета. И все это происходило на трансокеанских рейсах дальнего следования, обычно заполненных важными клиентами, постоянно пользующимися услугами этой авиакомпании. Сложно навскидку определить количество потерянных авиакомпанией денег, но я уверен, что речь шла о катастрофически огромных суммах.

Программное обеспечение в системах развлечений в полете функционировало безупречно, но его провал был обусловлен неумением адекватно взаимодействовать с людьми. Как могла

компания не спрогнозировать такой невеселый исход? Как упустила эту взаимосвязь? Цель данной книги – найти ответ на подобные вопросы и показать, как избегать таких катастроф современного высокотехнологичного мира.

Что будет, если скрестить компьютер и военный корабль?

Сентябрь 1997 года. Во время маневрирования в Атлантическом океане авианосец Военно-морских сил (ВМС) Соединенных Штатов Америки под названием *Yorktown*, являвшийся одним из новых крейсеров морфлота с боевой информационно-управляющей системой Aegis[2], неожиданно прекращает движение.

Во время калибровки топливного клапана техник ВМС ввел нуль в один из управляющих компьютеров, оснащенных процессором Pentium Pro и операционной системой Windows NT. Программа предприняла попытку поделить другое число на это нулевое значение, иначе говоря, выполнить действие, запрещенное в математике. Результатом стал глобальный сбой во всей системе управления бортом. Оставшись без компьютерного управления, двигатель встал, и корабль дрейфовал на волнах 2 часа 45 минут, до тех пор пока его не отбуксировали на базу. Удача, что подобное не случилось в зоне боевых действий.

Что будет, если скрестить компьютер и военный корабль? Адмирал Нимиц[3] перевернулся бы в своем гробу! Проигнорировав вышеописанную неудачу, ВМС решили оснастить компьютерными системами все имеющиеся корабли в целях экономии на человеческих ресурсах. Чтобы противостоять возражениям, они возложили всю вину за указанный инцидент на человеческий фактор. Ввиду того, что процесс проектирования программного обеспечения вышел из-под контроля, индустрия высоких технологий должна выбрать один из вариантов: навести порядок в этом процессе либо продолжать обвинять во всех грехах обычных пользователей, в то время как все более масштабные механизмы будут простаивать, намертво обездвиженные.

Технобезумие

Как-то в одном из номеров *Wall Street Journal* вышла статья, в которой был описан анонимный видеоклип, вирусно распространившийся через электронную почту. В этом клипе «…Усатый Обыватель в рубашке с коротким рукавом сгорбился над своим компьютером. Вдруг, в порыве гнева, он наносит удар по монитору. Удивленный коллега заглядывает в его отсек, а человек со всей силы швыряет в монитор клавиатуру, отчего тот валится на пол. Вскочив с места, безумец настигает поверженный монитор и наносит завершающий яростный удар».

Статья продолжалась рассуждениями о том, что данный клип «неимоверно всколыхнул» общественность и что он, по-видимому, вскрыл «мощную скрытую тенденцию к технобезумию».

Удивительная ирония этой ситуации в том, что для просмотра или простой пересылки этого видеоклипа нужно владеть компьютером на достаточном уровне. Человек на видео вполне мог быть актером, но ему удалось задеть струну, которая есть у каждого человека в бизнесе. Разочарование, вызванное сложностью и неприятием программных продуктов, нарастает стремительными темпами.

По закрытым спискам рассылки ходит шутка о «компьютерном синдроме Туретта». В основе этой игры слов лежит упоминание психического расстройства, похожего на нервный тик. Люди, страдающие таким расстройством, склонны бесконтрольно разражаться приступами ругани. Юмор состоит в том, что, окажись вы в помещениях практически любого современного офисного здания, вы услышите, как в целом нормальные люди, работая на своих персональных компьютерах, раздраженно стискивают зубы и непрестанно шепчут бранные слова. Одному богу известно, что вызывает в них такие приступы безумной ярости: утерянный документ, недоступное изображение или то самое взаимодействие, которое выводит из себя. А быть может, программа просто ненавязчиво удалила единственный имеющийся у пользователя экземпляр пятисотстраничного файла просто потому, что его ответом было «Да» в очередном диалоговом окне, похожем на то, где предлагается сохранить изменения, а на деле спрашивающем, желает ли он стереть данные.

Отказ индустрии нести ответственность

Мы живем в мире, который заполонили высокотехнологичные инструменты. Компьютеры безраздельно властвуют на наших рабочих местах и в наших домах, а транспортные средства пичкают гаджетами на кремниевых микросхемах. Каждое из этих компьютерных устройств невероятно мощное и оснащено богатым набором опций, но все они столь же грандиозно сложны и демонстрируют запутанное поведение.

Индустрия высоких технологий упрямо не признает тот простой факт, о котором мог бы поведать каждый владелец мобильного телефона или текстового процессора: *наши компьютеризированные устройства слишком сложны в применении*. Проектировщики программ и высокотехнологичных гаджетов гордятся своими достижениями. Разработчики программного обеспечения[4], также принимающие участие в их создании, прилагают все возможные усилия, чтобы упростить использование этих инструментов, и даже достигли в этом отношении некоторого успеха. Они думают, что имеющийся уровень простоты использования их продуктов настолько велик, насколько это позволяет их техническое устройство. Как истинные инженеры, они безгранично верят в технологии и имеют твердое убеждение, что только новейшие разработки – например, в области распознавания речи или искусственного интеллекта – способны улучшить опыт пользовательского взаимодействия.

Каким бы смешным это ни казалось, но в случае с программными продуктами *меньше всего* улучшений в отношении их использования принесут именно новейшие технологии. С точки зрения *технического устройства* сложный и запутанный программный продукт практически ничем не отличается от простой, мощной и приятной в использовании программы. Здесь все дело, скорее, заключается в уровне культуры, подготовки и отношении людей, которые создают эти программы, а отнюдь не в применяемых микросхемах и языках программирования. Червоточина находится в *процессе* разработки, а не в инструментарии.

Во главе индустрии высоких технологий каким-то странным образом оказались программисты и инженеры, чья сложная инженерная культура теперь превалирует в этой сфере. Топ-менеджеры фактически утратили способность влиять на индустрию высоких технологий, невзирая на занимаемые ими должности. Инженеры – вот кто правит бал. Стремясь наилучшим образом употребить все возможности, открывшиеся нам с изобретением кремниевой микросхемы, мы забыли об ответственности. *Психбольница оказалась руках пациентов.*

Когда психически больные завладевают сумасшедшим домом, у них нет четкого осознания причин собственных проблем. Глядя в зеркало, проще всего видеть только лучшее в себе и игнорировать недостатки. То же самое происходит и с разработчиками в тот момент, когда они пристально разглядывают результаты своего труда – они упускают из виду его плохие стороны. Вместо этого их взору предстает невероятная гибкость и мощь. Их взгляд направлен на богатство функционала и опций. Они упрямо отказываются признавать, что пользование таким продуктом вызывает страдания, требует множества мучительных часов для его изучения и что пользователи

сойдут с ума от безысходности и уничижения, будучи вынужденными применять его каждый день.

Что послужило причиной написания этой книги

Я занимался изобретением и разработкой программных продуктов двадцать пять лет. Долгие годы проблема запутанного программного обеспечения не давала мне покоя, пока, наконец, в 1992 году я не оставил программирование, чтобы полностью посвятить себя компаниям-разработчикам, помогая им делать их программные продукты приятнее и проще в применении. И тут произошло чудо! Ко мне вдруг пришло понимание, что только после того, как я освободил себя от программистских требований, я по-настоящему осознал, какими мощными они были. Программирование способно настолько захватить ваш ум своей сложностью, что только эта задача будет подавлять все прочие измышления, в том числе и заботу об удобстве пользователя. Я пришел к этой мысли лишь тогда, когда силой разорвал эти программистские узы.

Вместе с этим открытием ко мне пришло также осознание того, почему программы выглядят столь неудачными, по мнению пользователей. В 1995 году увидела свет моя книга[5], в которой я описал то, что узнал, и которая существенно повлияла на процесс разработки некоторых современных программ.

Чтобы стать хорошим программистом, необходимо обладать неким умением чувствовать природу и потребности компьютера. Вот только природа и потребности компьютера абсолютно чужды природе и потребностям людей, которые будут в конечном счете пользоваться этим компьютером. Процесс создания программ требует от разработчиков так много интеллектуальных усилий и такой степени вовлеченности в задачу, что они вынуждены полностью погружаться в инородный для человека образ мышления. В голове программиста все, что требуется для процесса разработки, получает наивысший приоритет, оставляя далеко позади какие-либо потребности внешних пользователей. Даже языки, на которых говорит каждый из этих миров, противоречат друг другу.

Процесс программирования и процесс разработки простых в использовании программ не согласуются по той причине, что программист и конечный пользователь преследуют в корне разные цели. У программиста есть потребность, чтобы разработка велась легко и непринужденно. А у конечного пользователя потребность звучит иначе – легким и непринужденным должен быть процесс взаимодействия с программой. Создать такую программу, которая удовлетворяла бы потребности и тех и других, не удается практически никогда. В современной индустрии компьютеров программисты наделены возможностями для создания дружелюбных пользователю взаимодействий, но из-за постоянного давления конфликта интересов они просто не в состоянии действительно сделать что-то.

Для процесса разработки программного обеспечения характерна такая особенность, что его результаты можно увидеть только по завершении всей работы. Отсюда вытекает тот факт, что любые прогнозы со стороны человека, не связанного с программированием, уже не смогут что-либо изменить. Использовать программное обеспечение для настольных компьютеров так тяжело потому, что в его создании принимают участие исключительно программисты и нет никого, кто выполнял бы роль посредника между программистами и пользователями. Изделия наподобие телефонов и камер всегда имели изрядную долю механических деталей, что позволяло быстрее изучать их устройство. Но исходя из наших исследований, если скрестить компьютер и любую другую вещь, характер компьютера непременно возобладает в ней.

Ключом к решению данной проблемы является проектирование пользовательского взаимодействия перед началом непосредственно процесса разработки. Нам нужно вырастить новый класс профессионалов пользовательского взаимодействия, которые будут проектировать поведение программ. Сегодняшние программисты осознанно подходят лишь к проектированию кода, а пользовательское взаимодействие отходит на второй план. Они планируют, что будет делать программа, но не то, как она будет себя вести, коммуницировать с пользователем и уведомлять его. Проектировщики взаимодействия, напротив, уделяют максимум внимания именно тому, как пользователи воспринимают программные интерфейсы различных устройств и как взаимодействуют с ними.

Искусство проектирования пользовательского взаимодействия достаточно молодое и знакомо не всем программистам. Но даже если программисты и признают его важность, они обычно фокусируются на нем лишь в завершение создания кода. Только в этом случае поздно что-либо менять.

Специалисты, управляющие проектами по разработке программ и устройств на их основе, либо попадают в заложники к программистам из-за своей недостаточной технической подкованности, либо ведут себя чрезмерно понимающе, так как сами являются разработчиками. Пользователям таких программ и устройств невдомек, что эти продукты могут также обладать необходимым функционалом и при этом обеспечивать приятное взаимодействие, как все прочие качественно спроектированные инструменты.

Программисты отнюдь не являются нашими врагами. Они прилагают массу усилий, чтобы придать своим программам должной легкости взаимодействия. К сожалению, они воспринимают все через призму собственного опыта; в итоге программами с легкостью могут пользоваться только другие разработчики, чего не скажешь о простых людях.

Невозможно даже представить стоимость некачественно спроектированного программного обеспечения. Сколько стоит потерянное время Джейн и Сунила, негодование скучающих пассажиров самолета, жизни людей, летевших роковым рейсом 965? Это нельзя измерить. Но наибольшую цену мы платим за упущенные возможности. Пока мы растрачиваем свое время на раздражение, попытки разобраться, разгребание последствий и несем издержки в смертельной схватке с программами, мы не используем весь тот заложенный в них потенциал обещаний быть более человечными, мощными и дружелюбными созданиями, какие только можно вообразить. На самом деле программное обеспечение невероятно гибкое и способно превзойти самые смелые ожидания. Все, что для этого нужно, – создать условия для разумного сотрудничества между проектировщиками пользовательского взаимодействия и разработчиками программ.

2. Когнитивное сопротивление

Одно дело – понимать, что проблема есть, а другое – предложить план ее решения. Одним из ключевых моментов при решении проблем является тот язык, который мы используем. За годы своей работы я изобрел множество удобных терминов и ментальных моделей. Они доказали свою актуальность при обозначении проблемы, вызванной сложными в применении программными продуктами. В этой главе вы познакомитесь с этими терминами и идеями и увидите, как с их помощью польза от проектирования взаимодействия проникнет в наш дефективный процесс разработки.

Поведение, не зависимое от физических сил

Едва оставив позади индустриальную эпоху, мы оказались в преддверии эры информации, но инструменты в наших руках все те же. В индустриальную эпоху инженерам была под силу любая задача. Подчиняя себе сталь и бетон, они воздвигали мосты и небоскребы, создавали автомобили и космические корабли, которые прекрасно работали и радовали тех, кто ими пользовался. Теперь мы осторожно ступаем в век информации, нам все чаще приходится работать с программами, и мы снова задействуем лучших из наших инженеров для решения задач. Но, в отличие от прошлого, все уже не так просто. Конечно, компьютеры обладают несомненной производительностью и скоростью, а на программное обеспечение в целом можно положиться, но теперь мы столкнулись с расстроенными, неудовлетворенными, несчастливыми и непродуктивными пользователями.

Сегодня инженеры обладают не меньшим потенциалом, чем они обладали ранее, поэтому отсюда следует, что перед ними предстала качественно новая проблема, отличная от всего того, с чем им приходилось сталкиваться в индустриальную эпоху. Будь это не так, они успешно решали бы задачи и прежними инструментами. За неимением подходящего термина я обозначил сущность этой новой проблемы как «когнитивное сопротивление». Это такое сопротивление, с которым сталкивается человеческий ум, когда ему приходится разбираться в сложной системе правил, меняющихся в зависимости от поставленной задачи. Процесс взаимодействия с программами обладает высоким коэффициентом когнитивного сопротивления. Процесс взаимодействия с физическими устройствами, даже сложными, обычно имеет более низкий коэффициент сопротивления; это объясняется тем, что у механических устройств, как правило,

ограниченный набор возможных состояний относительно количества возможных действий с ними.

Играть на скрипке чрезвычайно сложно, однако это действие обладает низким коэффициентом когнитивного сопротивления. Несмотря на то, что скрипач производит с ней множество различных по сложности манипуляций, извлекая самые невероятные звуки, скрипка никогда не войдет в метасостояние, при котором в зависимости от разных способов игры она будет звучать как труба или колокольчик. То, как поведет себя скрипка, всегда можно предвидеть: это подчиняется законам физики, хотя поначалу бывает не так просто понять ее поведение и научиться с ней обращаться. У микроволновой печи, напротив, высокий коэффициент сопротивления, так как каждая из десяти ее кнопок с цифрами может находиться в двух режимах, в зависимости от контекста. В одном случае кнопки регулируют интенсивность микроволнового излучения, а в другом они же отвечают за продолжительность тепловой обработки. Из-за таких переключений, в дополнение к отсутствию сенсорной обратной связи, по которой можно было бы определить, в каком режиме находится печь, когнитивное сопротивление возрастает в разы.

К примеру, клавиатура пишущей машинки не имеет метафункций. Нажимая клавишу с буквой «У», вы увидите на странице букву «У». Нажав последовательность клавиш из букв «СТЕРЕТЬ ВСЕ», вы увидите на листе бумаги слова «СТЕРЕТЬ ВСЕ». Компьютер же, в зависимости от контекста, при той же последовательности букв может задействовать метафункцию. Он выполнит операцию более высокого порядка и *в самом деле что-то сотрет*. То есть машина будет вести себя уже не так однозначно.

Когнитивное сопротивление, как и сопротивление в физическом мире, порой даже полезно, если присутствует в небольших количествах, но если оно увеличивается стремительными темпами, то вред от него растет экспоненциально. При этом обычное сопротивление является физической силой, которую можно выявить и измерить, а когнитивное сопротивление – инструмент в некотором смысле «судебный», и не нужно воспринимать его буквально. Но стоит отметить, что такие понятия, как любовь, честолюбие, мужество, страх, истина, при всей их реальности нельзя выявить и измерить. Равно как к ним нельзя применить методы инженерной науки.

Опытные производители микроволновых печей, как правило, привлекают экспертных консультантов по человеческому фактору, чтобы спроектировать такие кнопки, различать и нажимать которые пользователю будет удобно. Однако все, к чему адаптируют кнопки, эксперты по человеческому фактору, – это глаза и пальцы человека, но не его ум. В результате у микроволновых печей практически отсутствует коэффициент сопротивления в отношении физических действий с ними, зато когнитивное сопротивление при попытках разобраться в их поведении остается на высоком уровне. Открывать и закрывать дверцу, физически нажимать на кнопки достаточно легко, но вот добиться от панели управления выполнения нужных вам задач бывает не так просто. При всей сложности управления микроволновой печью мы в целом понимаем, как она работает, и это заставляет нас забыть о прочих трудностях взаимодействия с ней. Вспомните, как часто вы ошибочно устанавливали время разогрева на одну секунду или один час вместо одной минуты? Или как часто вы включали десятиминутный разогрев на пятом уровне мощности вместо пятиминутного на десятом уровне?

Каждый элемент на экране компьютера имеет свой коэффициент когнитивного сопротивления. Даже программы с таким простым интерфейсом, как браузеры, заставляют мозг пользователя работать в разы интенсивнее, чем при взаимодействии с любым механическим устройством. Это происходит оттого, что за каждой синей гиперссылкой скрывается переход к другой странице интернета. Все, что вам доступно, – это кликать по ссылке, но конечная точка, в которую вы при этом попадете, может измениться без каких-либо предупреждений, и вы не можете на это повлиять. В основе гиперссылки лежит метафункция. Из-за наличия подобных дополнительных «гиперсмыслов» объект и обзаводится когнитивным сопротивлением.

Дизайн и проектирование – понятия внушительные.

Суть этой книги в том, что в проектировании продуктов, обладающих интерактивностью, должны принимать участие проектировщики взаимодействия (interaction designers), а не разработчики программного обеспечения (software engineers). Программисты обычно сразу воспринимают это утверждение в штыки, ведь проектирование и дизайн составляют значительную часть их работы. Более того, их смертельно пугает тот факт, что теперь они будут лишены самого интересного и творческого аспекта процесса разработки и обречены писать нудный код, не получая от этого никакой радости. Это совершенно неверно. Их беспокойство произрастает из неясного понимания понятий «дизайн» и «проектирование».

Дизайн и проектирование присутствуют на протяжении всего процесса разработки программного продукта – от выбора языка программирования до подбора цвета фургона, доставляющего коробки с готовым продуктом. Ни в одном другом аспекте этого длительного и требующего полной отдачи процесса так не задействованы дизайн и проектирование, как в программировании. Программистам приходится принимать решения, связанные с дизайном и проектированием, на каждом шаге процесса разработки. Программисту нужно продумать, как осуществлять вызов одной процедуры через другую, как передавать, хранить, изменять и делать доступной информацию и сообщения о состоянии, как обеспечить валидность кода. Все эти решения, как и миллионы им подобных, – это непосредственно связанные с дизайном и проектированием решения, успешность которых зависит от того, насколько хорошо программисты используют свою рассудительность и мастерство.

Все многообразие видов дизайна и проектирования я могу легко поделить надвое. По одну сторону останутся те решения, которые непосредственно касаются взаимодействия с пользователем, а по другую – все, что к этому не относится. Когда вы встретите упоминание «проектирования взаимодействия» в этой книге, знайте, что под этим я понимаю только первую категорию решений. В отношении дизайна и проектирования, не влияющего на конечного пользователя, я применяю термин «проектирование программного продукта» (program design).

Осуществить такое разделение, основываясь исключительно на разнице в технических аспектах, представляется невозможным. Никак не получится выразить его в терминах, которыми привыкли оперировать инженеры, поскольку отличительный фактор здесь базируется на свойствах человеческой личности, а не на технических характеристиках, а инженерные правила неприменимы к людям. Например, проектировщики взаимодействия часто не придают значения выбору языка программирования для разработки программы. Тем не менее, как только от используемого языка будет зависеть время отклика программы, что, несомненно, относится к аспектам проектирования взаимодействия, проектировщику найдется что сказать.

Практически все вопросы проектирования взаимодействия лежат в плоскости выбора того, как будет вести себя программа, какой функциональностью обладать, как информировать пользователя и каким образом все это будет ему представлено. Процесс проектирования

взаимодействия конечного продукта – это единственное из того, что я хочу отнять у программистов и передать в руки компетентных проектировщиков.

Взаимосвязь программистов и проектировщиков

В технологическом мире инженеров главная роль была отдана проектированию архитектуры программного продукта, а о проектировании взаимодействия на благо конечного пользователя вспоминали лишь постфактум, когда оставалось время. Одна из целей, которую я преследую в этой книге, – обозначить преимущества от смены приоритетов и добиться того, чтобы проектирование взаимодействия начали считать первоочередной задачей в процессе создания программных продуктов.

Дизайн большинства программ возникает по воле случая

Проектирование тростниковых хижин и подземных ходов происходит не всегда осознанно, исходя из возможностей камня и тростника. То же можно сказать и о программном обеспечении – его устройство подчинено неведомым законам языков программирования и баз данных. Значительное влияние на все перечисленные способы проектирования оказывают сложившиеся устои. С той лишь разницей, что строитель-проектировщик хижины сам будет в ней жить, в то время как программисты обычно не являются теми, кто использует спроектированное ими ПО.

Реальное положение дел состоит в том, что компании, которые ведут разработку программ, не содержат в штате людей, хоть что-то понимающих в проектировании пользовательского взаимодействия. Однако те же самые несведущие люди весьма неплохо разбираются в проектировании программных продуктов и имеют собственное сложившееся представление о том, что нравится *лично им*. В итоге получается то, что получается: дизайн, спроектированный под них самих, при этом они выбирают лишь те аспекты, которые легко и приятно реализовать в коде, и воображают, будто на самом деле все это создается для пользователя. И хотя разработчики пребывают в уверенности, что в отношении дизайна и проектирования проделан большой пласт работы, речь здесь в большей степени идет только о *проектировании программного продукта*, и совсем немного выполняется для проектирования пользовательского взаимодействия.

Поскольку несовершенное проектирование – это тоже проектирование, получается, что тот, кто берет на себя принятие решений о поведении программы, выполняет роль проектировщика пользовательского взаимодействия. Маркетолог ли это, настаивающий на добавлении привлекательной для него функции, или программист, пытающийся реализовать в программе удобные для него опции, – все они «проектируют».

Отличие хорошего проектирования от того, что похоже на постройку тростниковых хижин, вовсе не в инструментах или приспособлениях, которые для этого применяются, а в тех побуждениях, что им движут. Для настоящего проектировщика взаимодействия важны те цели, которых пытается достичь пользователь. Мнимые проектировщики руководствуются широким кругом любых других случайных доводов в процессе принятия решения. Просто поразительно, что к ним могут относиться персональные предпочтения, мнение близких, боязнь неизвестности, рекомендации *Microsoft*, промахи коллег – все может сыграть решающую роль. Но чаще всего их решения основаны на том, что будет самым простым в реализации.

Проектирование «взаимодействия» и проектирование «интерфейса»

Я предпочитаю говорить *проектирование взаимодействия*, а не *проектирование интерфейса*, потому что термин «интерфейс» предполагает, что код находится по одну сторону, а люди – по другую. А посредине есть некая прослойка, которая передает сигналы между ними. Это выглядит так, будто только интерфейс отвечает за нужды пользователя. В результате подобного отделения дизайна на уровне интерфейса программисты начинают справедливо мыслить приблизительно так: «Я могу кодить как моей душе угодно, ведь все равно интерфейс будет прикручен в самом конце». Процесс дизайна и проектирования откладывается напоследок, когда становится совсем поздно.

С помощью дизайна и проектирования интерфейса можно только «приукрасить» то поведение, которое уже заложено в программе, – это все равно что одеть в костюм от Armani Атиллу, предводителя гуннов. Так, интерфейс программы для создания отчетов позволит убрать

ненужные границы таблиц с рядами цифр или устранить иные мешающие восприятию помехи, обозначить цветом важные показатели, дать визуально богатую обратную связь, когда пользователь будет щелкать по различным элементам. Несомненно, лучше такое поведение, чем вообще никакого, но для качественной системы этого отнюдь не достаточно. *Microsoft* затрачивает миллионы долларов на дизайн и проектирование интерфейсов, но их программные продукты так и не заслужили любви пользователей.

Поведенческое проектирование определяет, как должны себя вести и как взаимодействовать с пользователем элементы программного продукта. Если применить эту концепцию к тому же примеру с отчетными таблицами, то мы увидим, что поведенческое проектирование показывает, какими инструментами можно воспользоваться для работы с отчетом и как отразить в отчете средние и итоговые значения. Проектировщики взаимодействия начинают свою работу вовне, изучая цели, которые требуется достичь пользователю, но при этом постоянно контролируют, чтобы эти цели коррелировались с глобальными потребностями бизнеса, техническими возможностями системы и сопутствующими задачами.

Можно пойти еще дальше и затронуть область *концептуального проектирования*, которая исследует, что для пользователя является по-настоящему ценным и выходит на первый план. В примере с нашими таблицами концептуальное проектирование может показать, что работа над таблицей с цифрами лишь второстепенная задача для пользователя, – на самом же деле его целью является выявить тенденции и закономерности. Это означает, что вам следует задуматься не о создании инструмента формирования отчетности, а о проектировании средства отслеживания тенденций. Чтобы одновременно дать пользователям ощущение силы и удовольствия, проектировщикам взаимодействия следует изначально думать в *концептуальном* ключе, затем в *поведенческом* ключе и только потом задумываться об *интерфейсе*.

Что такого особенного в программных продуктах

Когнитивное сопротивление пробирается во все программные продукты, вне зависимости от простоты их внутреннего устройства, и оно же делает их куда как более сложными в использовании, в сравнении с их собратьями из эпохи механизмов. К примеру, заглянем, что лежит у меня в карманах: несколько монет, мой швейцарский армейский нож и мой автомобильный брелок. По ножу сразу видно, что это изобретение индустриальной эры: можно легко понять, из чего он состоит, как функционирует и что с ним нужно делать – вам достаточно произвести с ним пару быстрых манипуляций. Извлекая лезвие ножа, вы сможете оценить его острие и предположить, насколько оно подойдет для резки.

У ножа большой набор из шести лезвий, дополнительно прилагается зубочистка и пинцет. Для чего предназначен каждый из инструментов, мне ясно сразу. Легко, практически интуитивно я понимаю, что делать с ножом, благодаря тому, как он ровно ложится в мою руку и удобен для пальцев. Пользоваться этим ножом одно удовольствие.

Система бесключевого доступа на моем автомобильном брелоке – существо совсем другого толка. Она имеет всего две кнопки, поэтому на первый взгляд кажется проще в управлении, чем нож. Стоит обтекаемому черному кусочку пластика скользнуть в мои пальцы, как они совершенно естественно нащупывают две кнопки, чья функция предельно ясна: нужно нажать для активации. Хорошо, если бы было так, но за кнопками скрывается кремний, а не сталь, поэтому все не так просто, как кажется на первый взгляд.

Одно нажатие большой кнопки блокирует двери автомобиля и одновременно ставит его на сигнализацию. Если нажать второй раз – двери разблокируются, а сигнализация отключится. Еще одна кнопка на брелоке, чуть поменьше размером, подписана как Panic. Стоит нажать ее однократно, как автомобиль начинает тихонько переливаться трелями, и так продолжается несколько секунд. Стоит удержать нажатие чуть дольше, как тихие трели сменятся адским воем автомобильной сигнализации, то орущей, то щебечущей, то завывающей во все свои 100 децибел, возвещая на всю округу, что некий дурень вроде меня только что сотворил ужасающе глупую вещь. Хуже того, после срабатывания сигнализации кнопки маленького пластикового брелока прекращают на что-либо реагировать, так что он становится бесполезным. Единственное, как я могу положить конец этому громкому возвещению о моей явной глупости, – это дойти до моего страшно завывающего автомобиля, сгорая от стыда под взглядом каждого встречного, воспользоваться ключом для открытия двери со стороны водителя, затем вставить ключ в зажигание и повернуть его. При этом я действительно ощущаю себя болваном. Случись так, что мой автомобиль взломали и ограбили, я бы чувствовал себя подавленным и оскорбленным, но совершенно точно не болваном.

На страницах своей предыдущей книги я говорил о том, что главной целью всех, кто пользуется компьютерами, является не чувствовать себя глупыми. Я пошел еще дальше и высказал предположение, что качественные интерфейсы способны уберечь пользователя от того, чтобы не дернуть рычаг катапультирования, непонятно как оказавшийся среди постоянно используемых элементов панели управления. Рычаг катапульты – это классический пример устройства, из-за которого пользователи могут почувствовать себя глупыми, ведь он находится прямо перед ними. Если человек случайно потянет за этот рычаг, то поставит себя в весьма неудобное положение: это все равно что прийти утром в офис, забыв надеть брюки. С моим швейцарским армейским ножом проделать подобное в принципе не получится.

Я *затрудняюсь даже представить*, для чего мне могла бы понадобиться вторая кнопка, но еще более непонятно, почему создатели брелока упустили прекрасную возможность предоставить мне именно те функции, которые *в самом деле* желанны и необходимы[6].

Безусловно, я ценю, что моя машина оборудована сигнализацией, но случаются ситуации, и их достаточно много, когда мне требуется просто закрыть машину и не ставить ее на сигнализацию. Когда я на несколько минут заскакиваю в Starbucks за кофе, то не нуждаюсь в таком же уровне безопасности моего авто, как, например, в аэропорту. Мне действительно хотелось бы иметь возможность открывать и закрывать машину удаленно и не включать при этом сигнализацию. Это бы мне очень пригодилось, когда я езжу по местным магазинам или подкидываю детей до школы.

Еще одной, не менее полезной и желанной опцией была бы поддержка системы запирания автомобиля с более высоким уровнем надежности. Периодически, когда я возвращаюсь к своей машине, которую до этого закрывал, выясняется, что он уже не заперт и произошла эта перемена в мое отсутствие. Этот инцидент имеет место, когда рядом с моей машиной паркуется похожий автомобиль той же марки. Его обладатель жмет на кнопку, чтобы запереть свой автомобиль и одновременно, сам того не зная, подает сигнал разблокировки моей машине, то есть фактически отключает сигнализацию, оставляя мой автомобиль на растерзание социально опасным личностям. Беспокоит то, что подобная ситуация может возникнуть там, где для этого самые «благоприятные» условия: на больших парковках, какие есть, например, в аэропорту, я могу оставить машину на несколько часов или даже дней, предоставив ее потенциальному влиянию чужих систем дистанционной разблокировки. Технология была бы более полезной, если бы я мог блокировать и ставить автомобиль на сигнализацию с помощью нее таким надежным способом, чтобы открыть и разблокировать ее было возможно лишь при моем личном участии и применении металлического ключа, вставленного в замок. Разумеется, мне известно, что такой способ есть – потому что именно так можно выключить сработавшую сигнализацию. Но увы, те,

кто проектировал систему дистанционного управления, изрядно постарались, чтобы кто угодно мог снять мой автомобиль с сигнализации, несмотря на мои усилия обезопасить его.

Мой швейцарский армейский нож состоит из множества элементов и способен выполнять разные задачи, причем некоторые из его возможностей хитроумно спрятаны, тем не менее он довольно прост в изучении, предсказуем и интуитивно понятен. А вот система бесключевого доступа сложна в освоении, вызывает массу проблем и способна поставить меня в неловкое положение. Она не выполняет нужных мне задач и не позволяет контролировать собственный автомобиль и его сигнализацию на приемлемом уровне. Проще говоря, мне невыносимо взаимодействовать с этой системой. Она невообразимо плоха и вызывает во мне одну лишь ненависть.

Медведь-плясун

С другой стороны, появись передо мной возможность выбирать между ножом и системой бесключевого доступа, я в мгновение ока избавлюсь от ножа. Когда мне впервые довелось попробовать бесключевой доступ, я уже не представлял, как раньше мог обходиться *без него*. Это единственная наиболее удобная из всех опция в моем автомобиле, и я пользуюсь ею чаще остальных. В сравнении с ножом я применяю эту функцию в десять раз чаще. При всем ее непроработанном и неудобном устройстве она остается впечатляющим изобретением. Вообразите себе человека, который выводит на городскую площадь огромного медведя на цепи, собирает монетки со зрителей и заставляет медведя пуститься в пляс. Городские жители подтягиваются поближе, чтобы посмотреть на это невиданное зрелище – как большое неповоротливое животное неуклюже топчется на своих кривых лапах. Плясун из медведя никудышный, но зрителей *впечатляет не его танцевальное мастерство, а тот факт, что он в принципе совершает пляшущие телодвижения*.

Так же и с системой бесключевого доступа – впечатляет не то, что она работает без нареканий, а то, что она вообще хоть как-то работает. И тут я готов терпеливо сносить все проблемы взаимодействия, лишь бы иметь те выгоды, которые она дает мне относительно доступа к моей машине.

С открытием потенциала кремния мы получили столь изумительные возможности, что готовы с легкостью закрыть глаза на связанные с этим затраты. Оказавшись на необитаемом острове, вы вряд ли откажетесь от спасения, даже если к вам подплывет старая ржавая посудина, кишащая крысами. Между решением проблемы и отсутствием решения как такового настолько громадная разница, что мы готовы преодолевать сопряженные с этим трудности.

Сложность решения проблемы не в том, что нельзя разработать более удобное взаимодействие, а в том, что мы привыкли повсеместно соглашаться с использованием некачественных программ, принимая это как неизбежную данность. Завидев на горизонте ветхое проржавевшее спасательное судно, мы не задаем вопросов, какие там удобства, а радостно прыгаем на борт и довольствуемся тем, что имеем.

Экспертам в программном обеспечении приходится привыкать чувствовать себя комфортно при высоком уровне когнитивного сопротивления. Они с гордостью восхваляют свое умение выполнять рабочие задачи, несмотря на неудобства. У обычных же людей, кто только начинает пользоваться каким-либо программным продуктом, недостаточно профессиональных знаний, чтобы судить, как снизить уровень когнитивного сопротивления. Взамен они верят советам компьютерных умников, которые просто пожимают плечами и замечают, что для использования таких программных продуктов нужно обладать «компьютерной грамотностью». Программисты же, в свою очередь, винят во всем технологии, пытаясь объяснить пользователям, что проблемы с взаимодействием – это закономерное свойство всей техники и избежать этого не получится.

Но на самом деле это не так. Избежать сложных взаимодействий вполне возможно.

Источником когнитивного сопротивления являются не сами технологии, а те, кто управляет их разработкой. Они здесь главные, поскольку способны мыслить на языке кремниевых микросхем, и думаю, что этим свойством обладают и остальные. Они являют миру технологические артефакты, которые пытаются разговаривать с пользователем на том же языке, на котором их проектировали. Вместо того чтобы отделать салон автомобиля кожей и деревом, они предпочтут оставить на виду раскаленную сталь и скрежещущие механизмы. Будучи истинными инженерами, они больше задумываются о механизмах, нежели о кожаной отделке, потому и этот интерфейс выражается в терминах из области «реализации». Из-за этого я привык говорить, что такие продукты обладают *моделью реализации*.

Расплата за опции

Значительному количеству разработчиков программных продуктов неведомо, как создавать удобные в использовании программы, но вместо этого им хорошо известно, как пичкать их новыми опциями, потому они делают именно это.

Физические объекты, к которым относится мой швейцарский армейский нож, по естественным причинам не приветствуют разрастание новых опций, выходящих за пределы жизненной необходимости. Каждое новое лезвие или приспособление в моем ноже влечет для производителя дополнительные затраты. Создатель ножа отдает себе в этом отчет, поэтому каждая из вновь предлагаемых возможностей проходит этап строгого анализа на предмет необходимости до того, как будет реализована в конечном продукте. На языке инженеров это называется «отрицательной обратной связью» – то есть внутренние силы стремятся стабилизироваться и прийти в равновесие. Например, трение колес автомобиля о дорожное покрытие формирует отрицательную обратную связь в системе рулевого управления, так что, если вы отпустите рулевое колесо, оно будет стремиться вернуться к первоначальному положению.

В индустрии программного обеспечения превалирует иной подход. В связи с тем, что новые опции реализуются в будто бы неосязаемом программном коде, а не в физических материалах вроде стали, меди и пластика, у производителей, привыкших к прошлым условиям, создается впечатление, что опции достаются им почти даром. Им кажется, что программные продукты очень легко модифицировать, дополнять и внедрять в них «улучшения».

В этот самый момент я слушаю музыку Джимми Баффета с компакт-диска в моем компьютере. В небольшой программе-плеере с диска доступно множество разных опций: переход к предыдущему или следующему треку, выбор случайного трека, создание собственного плейлиста, воспроизведение в течение заданного времени, повтор воспроизведения, просмотр информации о Баффете в интернете, добавление альбома в избранное, внесение заметок к разным трекам, загрузка названий композиций из базы данных в сети, просмотр сведений о диске, создание списка избранных треков и много чего еще.

Все эти возможности, несомненно, приятны, и я бы не убирал их без лишней необходимости, но в общей своей массе они делают взаимодействие с программой крайне сложным и запутанным. Более того, когда мне нужно ответить на телефонный звонок и быстро поставить диск на паузу, я не могу найти нужную кнопку, потому что она погребена посреди всего этого огромного количества опций – бесплатных опций. Так что для меня они «бесплатными» отнюдь не являются. Какому-то незадачливому инженеру показалось, будто бы я мечтаю получить все эти бесплатные опции, однако мне просто нужен обычный плеер, в котором можно легко и быстро нажать на паузу.

Говоря о системе бесключевого доступа в моем автомобиле, я сильно сомневаюсь, что хоть кто-то из ее проектировщиков спросил себя: «Какие функции будут нужными и сколько их должно быть?» Напротив, я больше чем уверен, что какой-то младший инженер взял типовую микросхему, которая, по «счастливой» случайности, оказалась двухканальной. Один канал он приспособил для блокировки и разблокировки, а затем вспомнил, что у него есть еще один, лишний «бесплатный» канал. Этот инженер, вполне вероятно, будучи под началом инициативного менеджера по маркетингу – полного энтузиазма, но плохо информированного, – вообразил кажущуюся ему логичной схему, что выключение сигнализации вручную может для чего-то пригодиться. Наверное, он даже гордился тем, что смог предоставить дополнительную функциональность без явных затрат.

Вставить полноценный микропроцессор в ваш автомобильный брелок, микроволновую печь или мобильный телефон гораздо дешевле, нежели использовать отдельные микрочипы и электронные детали. Таким образом, новая технологическая экономика задает направление проектирования. Процесс добавления новых физических компонентов управления все еще сдерживается отрицательной обратной связью затрат на производство, но появление новых функций в программах не может сдержать уже ничего. Добавление новых программных опций представляется создателям программ делом незатратным, так что каждую предложенную опцию они считают хорошим вложением до тех пор, пока кто-то не сможет доказать обратное. В отсутствие регуляторов этого процесса продукты моментально обрастают множеством ненужных опций, что только усложняет процесс использования продукта для пользователя. Разумеется, эти функции представляются как совершенно необходимые преимущества, при этом, конечно, остается и основная, наиболее востребованная функция. Этот медведь уже начал приплясывать.

Что касается настольных компьютеров, то здесь недостаток обратной связи ослабляет их мощь. Разработчик программы воображает, что можно добавить какие его душе угодно функции и при этом они останутся «бесплатными», коль скоро их можно задействовать с помощью мыши и клавиатуры. В результате экраны заполоняют сотни непонятных значков, кнопок, элементов меню, каждый из которых должен реагировать на нажатие клавиш или на клик мыши. И как при этом пользователь должен выявить среди них незначительные опции и те, активация которых может повлечь серьезные негативные последствия?

Практически каждому коммерческому продукту свойственно усложняться с каждой последующей версией. Эволюционируя, программы получают новую функциональность и возможности, так что их интерфейс тоже обзаводится еще большим количеством элементов управления. В компьютерной прессе можно встретить такой термин, как «раздутое программное обеспечение» (bloatwear). Программы, подобные Lotus Notes, Adobe Photoshop, Intuit Quicken и Microsoft Word, так плотно набиты всевозможными опциями, что это сбивает пользователей с толку, тем более что лишь незначительная часть из них используется действительно эффективно, если вообще используется. А тем временем в этой массе всевозможных опций теряются действительно нужные и полезные.

Корпоративные программы страдают от этой проблемы еще больше, чем приложения для конечных пользователей. Компании, подобные *Oracle, PeopleSoft, ADP, SAP, Siebel*, разрабатывают комплексное программное обеспечение, необходимое для решения внутренних корпоративных задач. Такие программы невероятно сложно и непонятно устроены, а кроме того, перегружены различными опциями. С каждым ежегодным обновлением добавляются все новые и новые опции, а со старыми по-прежнему не представляется возможным разобраться тем, кто не пройдет многомесячное изнурительное обучение.

Поборники и потерпевшие

Медведи-плясуны заполонили мир вокруг нас. Компьютеры обещают нам такие впечатляющие возможности, что лишь немногие способны противостоять соблазну. Даже если у вас нет настольного компьютера, вы, несомненно, пользуетесь мобильным телефоном и банкоматами, а они работают на основе программного обеспечения. Нельзя просто так взять и не пользоваться компьютерами. Их цена не просто становится ниже – она стремительно падает, делая их доступными повсеместно. Множество вещей, которые мы привыкли считать механическими (или электронными), теперь обладают компьютерным разумом. Он присутствует

в автомобилях, стиральных машинах, телевизорах, пылесосах, термостатах, лифтах – и это далеко не полный список.

В то время как в индустриальную эпоху уровень пользы от различных устройств был прямо пропорционален сложности взаимодействия с ними, в новом информационном мире такая корреляция отсутствует, а наблюдается лишь то, что устройства быстро становятся все более сложными, а польза от них растет куда медленнее. Настроить старинный будильник с механической начинкой всегда считалось простой задачей. А проделать то же самое с современными программируемыми часами может оказаться сложнее, чем управлять автомобилем.

Из-за высокой степени когнитивного сопротивления общество оказывается расколотым надвое. Одни впадают в отчаяние и начинают считать себя глупыми от неудачных попыток взаимодействия, а другие, наоборот, испытывают мощнейший эмоциональный подъем от того, что смогли в очередной раз преодолеть все трудности. Такие сильные переживания превращают людей в «поборников» и «потерпевших». Для первых когнитивное сопротивление становится частью образа жизни, а вторых оно вынуждает уйти в подполье и видеть в компьютерах врага. Этот разрыв становится все заметнее.

*** * ***

Я называю первых поборниками, или апологетами, поскольку они из кожи вон лезут, чтобы оправдать свое восхищение медвежьими плясками. Словно партийные агитаторы в нелепых шляпах и с бестолковыми плакатами в руках, они восхваляют достоинства программ и бесстыдно замалчивают недостатки, как настоящие партизаны. В сущности, к данной категории можно отнести почти всех программистов, и это логично, учитывая их заинтересованность в этом деле. Удивительно другое – как много обычных пользователей, не связанных с технологиями профессионально, тем не менее находят оправдания тем, кто каждый день вынуждает их использовать продукт с отвратительным взаимодействием. Вот что говорят они в защиту своих истязателей: «Да это же просто. Только запомнить, что надо нажать вот эти две клавиши, а потом ввести корректное название в систему. Если я вдруг забуду это название, система позволит мне легко найти его». Им невдомек, что сама фраза «система позволит найти» звучит нелепо. Отчего бы системе самой не запомнить или не поискать данные? Апологеты – это такие люди, которые будут защищать компьютер лишь потому, что он помог им выполнить не выполнимые до этого времени задачи. Они указывают на медведя-плясуна и восторженно ахают: «Глядите, он в самом деле пляшет!»

Апологеты напоминают жертв стокгольмского синдрома. Так называют тех заложников, что испытывают симпатию к своим захватчикам, объявляя без тени иронии или здравого смысла: «Он такой чудесный человек. Он даже позволяет нам пользоваться туалетом».

Апологеты маскируются, называя себя «продвинутыми пользователями». Не имеет значения, насколько сложна в использовании система или как мало реальной пользы приносит функциональность, апологет будет неизменно восхвалять лишь мощность и широкий набор опций очередного гаджета, словно не замечая сложностей в его реальном применении.

От одной из моих коллег, работающей в сфере мобильных телефонов, я услышал фразу о том, что инженеры делают использование телефонов невероятно сложным, добавляя в них множество опций, которые используются крайне редко. Она назвала мобильные телефоны «мочащимися собаками». На мою просьбу пояснить, что она имеет в виду, она ответила: «Нужно слишком сильно любить собаку, которая постоянно мочится, чтобы везде таскать ее с собой».

Поражает, как удается компьютерам притягивать к себе такое огромное количество невероятно умных и высокоорганизованных людей. Те же самые люди, вероятно, увлекаются рискованными и опасными занятиями и видами спорта: горными лыжами, пилотированием, плаванием с аквалангом, игрой на бирже, альпинизмом. Участники всех этих занятий должны быть тщательно подготовлены, и даже самая крошечная невнимательность может повлечь неприятные последствия. Однако не будь эти занятия наполнены такой притягательностью, ощущением вызова, разве в этом случае их поклонники не остались бы дома у телевизора?

Сложность – это именно то, чем они так невероятно привлекают к себе. Это испытание духа, преодоление чрезвычайно трудной ситуации, в которой нельзя ошибиться. Только представьте себе, как измотанный альпинист, с которого градом катится пот, залпом выпивает банку изотоника Gatorade, чтобы восстановить утраченные силы, и с ухмылкой поясняет: «Да, парни, последний отрезок подъема был почти отвесным, у меня даже мышцы судорогой свело, пока я туда карабкался». Он *получает удовольствие* от подобных трудностей! Чем тяжелее испытание, тем больше удовольствия! Вот поэтому он и занимается альпинизмом!

Компьютеры оказывают на людей аналогичное вдохновляющее действие, ведь для взаимодействия с ними нужно преодолевать подобные тяжелые, безжалостные вызовы. Если вы недостаточно тренированы, чтобы постоянно оставаться на высоте, компьютер просто бросит вас лежать в пыли и жаловаться на судьбу. Только представьте себе измотанного программиста, который залпом выпивает банку кока-колы и с ухмылкой поясняет: «Да, парни, я еле разобрался! Этот сбой был вызван дурной логикой программы, причем это происходило, только когда куча превышала размер в 64 мегабайта, а иначе кеш просто не был задействован!» Для него в этом особое *удовольствие*!

Отсюда и возникают апологеты. Им доставляет наслаждение преодолевать тяжелые испытания и суровые препоны судьбы. Они счастливы работать в таких обстоятельствах, где за счет своих невероятных способностей они выделяются среди других. Альпинист становится апологетом, защищающим сложные и крутые горные подъемы. Компьютерный энтузиаст становится апологетом, защищающим неявное и сложное в управлении компьютерное обеспечение.

* * *

Другой фронт этого противостояния составляют потерпевшие. Они смутно осознают, что происходит нечто совершенно неправильное, но вот что именно – сказать не могут. Они не слишком хорошо разбираются в технике и взаимодействии с ней, но понимают, что проблема имеет место. Они лишь могут определить, что для них *сложно*, а что *легко*, и в таком понимании *компьютеры – это сложно*.

Но они, как и другие люди, не могут просто перестать использовать компьютеры – техника необходима для решения насущных рабочих задач. Стиснув зубы, они вынуждены смириться с поведением программного обеспечения, похожего на медведей-плясунов. Им в голову не приходит, что компьютеры способны вести себя иначе; им лишь известно, что каждый раз при взаимодействии с ПК они начинают ощущать себя менее уверенно, нежели в обычном состоянии. Словно крестьяне феодального строя Средних веков, они не могут изменить положение вещей, равно как и осознать, чего их лишают. Единственное, что они четко понимают, – это что их подавляют.

Апологетам свойственна такая фраза: «Поглядите, что я могу сделать на этом компьютере». Потерпевшие обычно говорят: «Наверное, я просто недостаточно умный, чтобы разобраться с этими новомодными технологиями». Апологеты произносят: «Смотрите, медведь-плясун!» Потерпевшие произносят: «Хотел бы я найти нечто, что способно плясать, и, видимо, медведь – это лучшее из всего, что я могу увидеть». К потерпевшим относится большая часть людей, кого не поражает та мощь, которую они обретают с приходом в их жизнь компьютера, а, напротив, их поражает тот факт, насколько глупыми они начинают себя чувствовать при этом.

Несомненно, все представители компьютерной индустрии, равно как и смежных индустрий, занимающихся созданием продуктов и предоставлением услуг, связанных с технологиями, принимают сторону апологетов. То, как они действуют, отражает то, как они видят мир. Они всегда готовы защищать свои изобретения, восхваляя их функциональность и производительность. А когда их спрашивают о взаимодействиях с человеком, они, словно политики, избегают отвечать на провокационные вопросы, вместо этого начиная расхваливать широкие возможности продукта, количество его новых опций и то, как много людей уже им пользуется. Они игнорируют плохое качество медвежьей пляски, но выставляют напоказ сам факт этого явления.

С быстрым повсеместным распространением интернета и упрощением доступа к нему технологический мир накрыла волна новоявленных поборников и потерпевших. Поборники восхищенно твердят, как много всевозможных сервисов и ресурсов теперь доступно через интернет, в то время как потерпевшие пристально вглядываются в экраны своих компьютеров, пытаясь сообразить, как найти что-то, представляющее для них пользу. Они вынуждены бесконечно долго ждать, пока очередной сайт загрузит массу ненужных изображений, и, даже дождавшись, теряются в ворохе сложной структуры переходов к бесполезной информации.

Всемирная паутина – это, наверное, самый крупный пляшущий медведь, с которым когда-либо приходилось сталкиваться человечеству.

Как мы реагируем на когнитивное сопротивление

Реакция подавляющего числа людей на когнитивное сопротивление в большинстве своем одинакова. Они довольствуются самым малым из всего многообразия, что им доступно, и напрочь забывают про остальное. Каждый пользователь осваивает минимальный набор функций для решения базовых задач, а все прочие просто-напросто игнорирует. Апологеты восторженно расписывают свои наручные часы, которые обладают функцией синхронизации с календарем на компьютере, да только ненавязчиво упускают тот факт, что последний раз эта функция пригодилась им полгода назад. Если вы укажете им на этот нюанс, они станут яростно защищаться – но на то они и апологеты.

Мой домашний центр развлечений обладает превеликим множеством разных функций – кажется, их в нем около тысячи. Я не апологет, но определенно помешан на всяких гаджетах. Я сумел освоить несколько любезно предложенных мне возможностей центра развлечений, но пользоваться ими эффективно довольно сложно. Например, у моего телевизора есть опция «картинка в картинке» (КвК) – это выглядит как небольшой дополнительный экран в правом нижнем углу основного экрана, отображающий вещание с другого канала. Функция эта полностью программная, и ею можно управлять кнопками на пульте. Теоретически такая опция могла бы пригодиться, например, чтобы следить за футбольным матчем на маленьком экране, пока на большом идет кинофильм. После того как продавец показал мне эту функцию в магазине, мне показалось, что это удобно.

Проблема в том, что использовать эту опцию невероятно сложно. Коэффициент когнитивного сопротивления очень высок, и я не могу освоить ее на достаточном уровне, чтобы ощутить какую-то выгоду. Куда как приятнее смотреть на один экран – как в старые добрые времена, когда телевизоры не могли предложить иного. Остальные члены моей семьи не захотели и один раз попробовать эту функцию и включают ее разве что случайно. Бывает, я возвращаюсь домой и вижу, как кто-то смотрит телевизор с КвК. Завидев меня в комнате, домашние сразу же требуют отключить эту опцию.

Диагональ экрана моего телевизора составляет 55 дюймов, а еще он оснащен звуковой системой Dolby Digital и принимает сигнал со спутника, но используется он мной и моей семьей *точно так же*, как наш старый телевизор Motorola с экраном в 19 дюймов в далеком 1957 году. Все его новомодные опции нам без надобности.

Предсказать, какие функции каждого нового устройства будут востребованы, а какие – нет, несложно. Чем больше манипуляций потребуется сделать, чтобы воспользоваться той или иной опцией, тем меньше она будет востребована, то есть наблюдается обратно пропорциональная зависимость. Так, экран моего нового телевизора, больший по размеру, более яркий и четкий, не притязает на дополнительные действия с моей стороны, поэтому он задействован на 100 %, пока телевизор включен, и я им весьма доволен. Спутниковое телевидение – очень желанный пляшущий медведь, поэтому я смиренно выношу все трудности, связанные с переключением источника сигнала, и смотрю спутниковые каналы приблизительно один раз в неделю или около того. Больше никто из членов моей семьи не смог совладать со спутниковой системой – пришлось сделать список подсказок с нужными для включения спутниковых каналов рычажками, кнопками и настройками, заламинировать его и поместить на кофейный столик. Что касается функции КвК – для ее включения и настройки используется около десятка разных кнопок, да еще к тому же не всегда понятно, как с ней взаимодействовать, а ее поведение не отличается привлекательностью. Промучившись с ней недолго, я перестал ее включать, а остальные последовали моему примеру.

Такие последствия когнитивного сопротивления, выраженные в отказе от использования функциональности, можно наблюдать практически в каждом офисе или доме почти с каждым программным продуктом или устройством на основе ПО.

Демократизация потребительского влияния

В обществе так сложилось, что чем сложнее был устроен тот или иной механизм, тем более высокопрофессиональные операторы требовались для взаимодействия с ним. Масштабные агрегаты всегда скрывали от посторонних глаз и подпускали к ним только особым образом обученных специалистов в форменной одежде. С приходом эры информации пришли и перемены, и теперь мы ждем от любителей способности совладать с технологиями, в разы более сложными, чем когда-либо видели наши предшественники.

Все больше и больше наших инструментов обзаводятся кремниевым разумом и все больше из них оказываются в руках неопытных пользователей. Двадцать пять лет назад обработкой междугородних вызовов занимался специально обученный оператор по устному запросу. В настоящее время самые сложные междугородние вызовы совершаются любым неподготовленным человеком самостоятельно, по нажатию на кнопки.

Несколько десятилетий назад даже на автозаправочных станциях присутствовали обученные сервисные специалисты. Сегодня каждый должен сам уметь пользоваться услугами заправки – не только заправить свой автомобиль, но и оплатить услугу кредитной или дебетовой картой. Двадцать лет назад взаимодействие с банком производилось не иначе, как с помощью отдельного обученного операциониста. Сегодня же вы самостоятельно коммуницируете с банком посредством терминала на заправочной станции или через банкомат.

Процесс инженерного проектирования при создании сложной системы не делает разницы между тем, будет ли ею оперировать специально нанятый профессионал или же неподготовленный новичок, который на это не подписывался. Процесс инженерного проектирования не обладает представлениями, как работать с этими особенностями человеческого характера. Он концентрируется только на вопросах реализации продукта: «из чего это будет сделано?», «как это будет сконструировано?», «какие управляющие элементы нужны для ввода всех возможных вариантов данных?».

Во всем виноват пользователь

Так как программы используются по большей части для решения рабочих задач, жертвы некачественного взаимодействия получают за свои мучения деньги. Им приходится задействовать программы в работе, так что не в их силах от них *отказаться* – в их силах только молча терпеть, покуда это возможно. Они вынуждены скрывать свое раздражение и подавлять то чувство собственной глупости, которое пытается внушить им программа.

Долгие годы я смотрел на десятки руководителей в сфере разработки программного обеспечения, когда они рисовали передо мной один и тот же схематичный рисунок, отражающий их видение рынка высоких технологий. На рисунке представлена пирамида (некоторые изображают ее перевернутой), состоящая из трех уровней, каждый из которых обозначен одной, казалось бы, безобидной фразой. Поверх этой пирамиды каждый менеджер непременно рисует некую шарообразную фигуру, призванную продемонстрировать сегмент рынка, на завоевание которого нацелилась компания. Но каждый уровень пирамиды на деле обозначен эвфемизмом – фразой, несущей завуалированный оскорбительный смысл, из тех, которыми пользуются фанатики, чтобы изгнать непосвященного из тайного общества.

Вот они, эти три эвфемизма, обозначающих разные типы пользователей: «неопытный», «компьютерно грамотный» и «продвинутый».

«Неопытный пользователь» – это, в понимании компьютерной индустрии, все равно что «недалекий и некомпетентный». На самом же деле эти люди таковыми не являются – их вынуждает чувствовать себя глупыми некачественное проектирование взаимодействия. Поборники программного обеспечения не считают таких пользователей за пользователей вообще, однако это не согласуется со здравым смыслом. Только с чего бы поставщику программных продуктов пренебрегать этой львиной долей рынка ПО? Все дело в том, что такая позиция снимает ответственность за провал с менеджеров и программистов и перекладывает бремя этой вины на плечи неопытных пользователей.

Пирамида эвфемизмов

Под оборотом «компьютерно грамотный пользователь» на самом деле подразумевается человек, которому причиняли боль столько раз, что он уже оброс толстой кожей и ничего не чувствует. Обладать компьютерной грамотностью – значит, что если вы вдруг потеряете документ, как Джейн из главы 1 «Загадки информационной эры», то уже не станете паниковать, а просто молча, медленно и методично начнете совершать бесполезный поиск файла в структуре файлов и каталогов, ни разу не пожаловавшись на это. Обучение компьютерной грамотности похоже на анестезию: пользователь плавно впадает в состояние полной бессознательности. В самом деле, зачем постоянно жаловаться на свою несостоятельность или ругать программу, без которой не обойтись при решении рабочих задач. Значительная часть пользователей даже не осознает, как много усилий они тратят на то, чтобы обойти несовершенства программных продуктов.

Апологеты в большинстве своем принимают компьютерную грамотность за некую награду, признак высоких достижений, вроде медали за меткость в стрельбе. Хотя куда более уместна здесь будет аналогия с Пурпурным сердцем – официальной наградой пострадавших от ранения в бою.

Продвинутые пользователи – те же апологеты. Это увлеченные технари, которые сумели преодолеть свои самые сильные инстинкты и превратились в успешных потребителей

программных продуктов с высоким коэффициентом когнитивного сопротивления. Они с гордостью преодолевают такие испытания, будто пытаются влезть на скалу в Йосемитском национальном парке.

Софтверный апартеид

В Голливуде бродит старая шутка про то, что, если внезапно спросить у любого посетителя в продуктовом магазине, как у него дела со сценарием, то он, не моргнув глазом, ответит: «Прекрасно! Я как раз переделал второй акт, чтобы придать действию драматизма!» Таким же образом теперь можно шутить и про Кремниевую долину. Стоит вам выцепить из очереди в Starbucks какую-нибудь незнакомку и задать ей вопрос о том, как дела с ее сайтом, она без промедления ответит: «Прекрасно! Я как раз переделала фреймы, чтобы оптимизировать навигацию!»

Обитатели Долины забывают, как сильно они не похожи на всех остальных, так что им стоит почаще напоминать себе об этом. Среднего пользователя компьютера в этих краях далеко нельзя назвать средним в понимании обычных людей.

Программисты обычно проводят время в среде таких же технарей, как они сами, работая в анклавах вроде Кремниевой долины, 128-й окружной автомагистрали в пригороде Бостона, Исследовательского треугольника в Северной Каролине, Редмонда, штат Вашингтон, и Остина, штат Техас. Создатели программ постоянно видятся с себе подобными, когда ходят по магазинам, обедают, отвозят детей в школу, отдыхают, а с недовольными пользователями ПК они практически не пересекаются. Более того, приступы раздражения, периодически возникающие среди пользователей, неясно в чей адрес, практически теряются в множественных проявлениях энтузиазма среди интеллектуальной элиты. Мы забыли, насколько велик разрыв между нами и другими жителями страны (если не сказать – мира), в неспособности использовать интерактивные технологии без раздражения.

Мы – те, кто находится в самом сердце индустрии, – провозглашаем понятие «компьютерная грамотность», подразумевая, что людям для того, чтобы пользоваться компьютером, требуется овладеть базовыми знаниями и навыками. Нам это требование кажется несложным и даже разумным. Мы думаем, что изучить основы – не такое уж большое усилие для пользователей, в обмен на радость от всех тех преимуществ, которые дают компьютеры. Но на самом деле мы требуем от них *слишком* многого. Наличие фундамента пользователей, обладающих компьютерной грамотностью, безусловно, значительно упрощает процесс разработки ПО, но также и тормозит развитие и возникновение успеха в индустрии и обществе. Апологеты могли бы возразить, что перед тем, как сесть за руль автомобиля, люди обычно учат правила дорожного движения и берут практические уроки вождения, но они упускают из виду тот факт, что ошибка на дороге может стоить кому-то жизни, а работа с программным обеспечением имеет меньшую тяжесть последствий. Если бы машины не могли привести к гибели, процесс обучения вождению выглядел бы так же, как иные осваивают Excel.

Когда мы оперируем понятием компьютерной грамотности, возникает другой, гораздо более коварный эффект. Между обеспеченными людьми и малоимущими появляется демаркационная полоса. Там, где в Америке требуется знание компьютера, чтобы преуспеть на рынке труда и стать чем-то большим, нежели продавец гамбургеров, трудности, связанные с изучением интерактивных систем, удерживают многих людей от попыток получить более интересную, солидную и высокооплачиваемую работу.

Пользователей не должны вынуждать становиться компьютерно грамотными, чтобы использовать компьютер для выполнения простейших ежедневных рутинных задач. Пользователь не обязан обладать цифровым чутьем, чтобы суметь совладать с видеомагнитофоном, микроволновой печью или электронной почтой. Более того, нельзя заставлять пользователей дополнительно получать компьютерное образование для решения задач в предметной области, в которой они уже и так являются специалистами. Например, бухгалтер, уже изучивший принципы бухгалтерского учета, не должен в дополнение учиться специальным навыкам работы с компьютером, чтобы использовать его для ежедневных задач. Ему должно быть достаточно профессиональных предметных знаний.

Чем больше экономика нашего общества уходит корнями в использование информационных технологий, тем более разделенным становится наше общество, хотя мы сами того не осознаем в

полной мере. Наверху оказываются люди, познавшие нюансы отличия оперативной памяти от жесткого диска. А низшим классом считаются те, кто полагает, что знание этих отличий несущественно. Ирония состоит в том, что знать, в чем тут разница, действительно *несущественно*для всех, кроме небольшого числа инженеров по аппаратному обеспечению. Однако значительное количество программных продуктов заставляет пользователей разбираться с файловой системой, где успех в выполнении ими рабочих задач зависит от осознания разницы между оперативной памятью и жестким диском.

Таким образом, понятие «компьютерная грамотность» представляет собой эвфемизм для обозначения социального и экономического апартеида. «Компьютерная грамотность» – это кодовое слово, по которому одна часть общества отделяется от другой.

А что же происходит с теми, кто не хочет становиться пособниками технократов и не намерен овладевать компьютерной грамотностью? Такие люди, в большинстве своем по собственному выбору, а в меньшинстве – по воле обстоятельств, вынуждены оставаться позади информационной революции. Многие технологические компании, например, отказывают кандидатам, у которых нет электронного почтового адреса. Больше чем уверен, что есть еще множество квалифицированных специалистов в своей области, кто не может устроиться на работу из-за того, что они не имеют доступа к сети интернет. Несмотря на восхваления апологетов, работа с электронной почтой на достаточном уровне эффективности сложна и требует определенного уровня компьютерной грамотности. Получается, что часть потенциальных работников отпадает. Это напоминает практику «красной черты», которую используют некоторые банки. Это незаконная процедура, и заключается она в том, что все дома какого-либо района по умолчанию принимают как неподходящие для обеспечения жилищной ссуды. И хотя красными линиями на карте якобы чертят по границам экономических областей, на самом деле в них явно прослеживаются расовые разграничения. Банковские служащие сопротивляются тому, что их называют расистами, но результаты говорят об обратном.

Программисты, которые твердят нам про компьютерную грамотность, тоже проводят красную черту, отделяя этнические группы, но мало кто видит это. Представляется очень сложным увидеть то, что на самом деле происходит, так как суть проблемы виртуозно скрывается баснями из сферы технологий. Выявить, что банковский служащий может выдавать ссуду под любой дом, достаточно легко. Но уже не так просто понять, что разработчик способен создавать программные продукты, взаимодействие с которыми будет приятно для пользователя даже из низшего социального слоя.

В нашей индустрии не принято признавать проблему отсутствия программных продуктов с качественным взаимодействием. Вокруг слишком много поборников, восторгающихся пляшущими медведями. Своими нарочитыми восторгами они душат на корню едва возникающие сомнения других в эффективности программных продуктов. До того как начать исследовать возможные решения, мы должны единогласно прийти в себя и осознать, насколько масштабна и злободневна эта проблема. Это цель нашего следующей части.

Часть II. Цена слишком высока

3. Пустые растраты

Не так-то просто, хотя иногда кажется иначе, впустую потратить миллионы долларов, но прекрасным подспорьем в этом нелегком деле может стать некачественный процесс разработки. Все дело в том, что процессу разработки не хватает одной маленькой ключевой детали: понимания, в какой момент считать продукт «завершенным». Без этого жизненно важного знания мы слепо полагаемся на произвольный дедлайн. Мы стремимся пересечь финишную черту и готовы затрачивать миллионы, лишь бы сделать это поскорее, но в очередной раз обнаруживаем, что этот финиш лишь мираж. В этой главе я постараюсь развенчать миф о следовании таким дедлайнам, что в итоге обходится слишком дорого.

Управление дедлайнами

В Кремниевой долине сложились некоторые навязчивые устои касательно времени вывода продуктов на рынок. Существует распространенное предположение, что выпустить продукт *прямо сейчас* гораздо лучше, нежели сделать это позже. Такая устоявшаяся необходимость служит оправданием нереалистичным амбициозным срокам сдачи и выгоранию

сотрудников, занятых в проекте, но на деле это лишь прикрытие, отвлекающий маневр, за которым скрываются намного более серьезные и глубокие страхи. Любому предпринимателю прекрасно известно, что вывести на рынок раздражающий и расстраивающий пользователя продукт спустя три месяца от начала его разработки *ничуть не лучше*, чем выпустить приятный пользователю продукт спустя шесть месяцев.

У каждого руководителя внутри есть как минимум два схожих страха. Руководители беспокоятся о том, в какие сроки их программисты смогут завершить разработку, и сомневаются насчет того, будет ли разработанный продукт достаточно хорош, чтобы в конечном счете стать успешным на рынке. Оба этих страха исходят от присущей каждому руководителю недостаточной ясности видения, что должен собой представлять конечный продукт. Все, что они могут про него сказать, – это что продукт «должен работать на компьютере пользователя» и «он не должен приводить к сбоям». Ввиду отсутствия такого видения они не могут оценить степень завершенности продукта.

В результате таких опасений под описание продукта, который «не приводит к сбоям», подходит и тот, который разрабатывался три месяца, и тот, разработка которого занимает шесть месяцев, с той лишь разницей, что на последний требуется три лишних месяца программирования и дополнительные чудовищные затраты средств. Стоит программистам начать работу, деньги улетают стремительно. Следуя такой логике, руководитель разработки считает самым важным приступить к программированию как можно раньше и закончить тоже как можно раньше.

Сознательный руководитель разработки быстро набирает программистов в команду и велит им незамедлительно приступать к работе. Он решительно отводит на завершение проекта всего несколько месяцев с даты его начала, и команда начинает сломя голову мчаться к финишной прямой. Однако при отсутствии должного проектирования проекта оба страха руководителя остаются в силе. Понравится ли продукт потребителям – все еще не ясно, так что его успешность на рынке покрыта мраком. И так же не ясно, как должен выглядеть готовый продукт, что покрывает мраком и степень его завершенности. Чуть позже в этой книге я продемонстрирую, как проектирование взаимодействия позволяет сгладить эти проблемы. А прямо сейчас я покажу, как основательно дедлайн способен пошатнуть процесс разработки до такой степени, что худшие опасения руководителя начнут сбываться.

Как выглядит «завершенный» продукт?

Когда у нас на руках окажется детализированное описание того, как должно выглядеть готовое программное обеспечение, появится возможность сравнить наше творение с указанным описанием и получить *полное представление*, завершен ли продукт.

Описания бывают двух типов. Мы можем либо подробно описать физический вид требуемого продукта, либо обозначить, какова будет ожидаемая реакция пользователя на конечный продукт. Так, в сфере строительства и архитектуры план является таким документом, который удовлетворяет условиям первого типа описаний. Если же речь идет о съемке нового фильма или открытии ресторана, в своем описании мы фокусируемся на тех эмоциях и ощущениях, которые должен испытать посетитель. Проектируя программы и продукты на их основе, нам обязательно следует прибегнуть к совмещению обоих типов описаний.

К несчастью, у многих программных продуктов такие описания просто-напросто *отсутствуют*. Вместо этого у них есть обширные перечни функций, сравнимые со списком покупок в магазине. Однако набить корзину для покупок мукой, сахаром, молоком и яйцами – это совсем не то же самое, что испечь торт. Торт получится только в результате следования всем шагам рецепта, и то, что мы приготовим в итоге, будет выглядеть как торт и обладать знакомым нам ароматом и вкусом торта.

Мнимый пекарь, у которого есть все ингредиенты для торта, но нет нужных знаний, как этот торт испечь, только впустую проведет время на кухне и получит сомнительный результат. Потребуй мы, чтобы торт был готов к шести часам, добропорядочный пекарь выполнит задачу точно в срок, но будет ли то, что он принесет, тортом? Все, в чем мы можем быть уверены, – что продукт появится вовремя, но вот о его успешности можно только гадать.

В большинстве строительных проектов мы способны точно оценить степень готовности объекта, потому что понимаем, как выглядит завершенный проект. Мы знаем, что здание готово, потому что оно выглядит и функционирует в точности так, как это описано на планах. Если объект должен быть сдан первого июня, это не всегда означает, что в указанный срок здание будет готово. Относительная степень готовности здания может быть оценена только в сравнении текущего объекта с планом.

Без подобных планов создатели программ оказываются лишенными ясного понимания, что считать готовым продуктом, поэтому они назначают приблизительную дату завершения, а когда она наступает, просто объявляют продукт готовым. Наступило первое июня? Значит, продукт готов. «Запускаем!» – говорят они, и дедлайн становится единственной контрольной точкой степени завершенности продукта.

Разумеется, программисты и предприниматели далеко не дураки, оттого продукт не получится слишком уж оторванным от реальности. У него будет широкий набор функций, и работать он станет стабильно и без сбоев. Он даже начнет прекрасно справляться с задачами, если окажется в руках людей, *сильно заинтересованных* в том, чтобы так оно и было. Может статься даже так, что продукт проходил юзабилити-тестирование, при котором группа добровольцев пытается выполнять на нем задачи под наблюдением специалистов по юзабилити[7]. Но так как количество предпринятых разумных мер предосторожности этим ограничивается, мы все еще не можем с большой долей вероятности ответить на вопрос, будет ли продукт успешным.

Закон Паркинсона

Руководители знают, что разработка программного обеспечения идет в соответствии с законом Паркинсона: работа заполняет все время, отпущенное на нее, увеличиваясь в объеме. Если вы работаете в сфере разработки ПО, то вам, скорее всего, известно и следствие из этого закона, которое называют «правилом 90/90» (автором этого правила считают Тома Каргилла из *Bell Labs*): «Первые 90 % кода занимают первые 90 % времени разработки. Остальные 10 % кода занимают вторые 90 % времени разработки». Это уничижительное правило просто-напросто говорит, что даже в тот момент, когда первые 90 % кода написаны, программистам *по-прежнему* неизвестно, в какой стадии находится проект! Руководство прекрасно знает, что сдать проект в срок не получится, какие бы сроки ни были установлены. А учитывая то, что программисты наиболее продуктивно работают под давлением, руководители применяют дедлайн как один из рычагов воздействия на них.

В 1980-е и 1990-е годы вице-президентом по разработке в небольшой, но достаточно влиятельной компании под названием *T/Maker* был Ройял Фаррос. Он говорил так: «Многие из нас устанавливали настолько жесткие дедлайны, что в них было *заведомо* невозможно уложиться. Такая ситуация лишний раз подтверждала одно из следствий закона Паркинсона: „Этап завершения проекта разработки требует в два раза больше времени, чем планировалось“. Я

пребывал в *твердом* убеждении, что, если установить дедлайн, например, через шесть месяцев, на самом деле это займет год. Таким образом, если нужно было получить продукт через два года, дедлайн надо было устанавливать на один год. Такие сроки просто огорошивали, но метод срабатывал всегда».

Риджли Эверс, предприниматель в сфере разработки программного обеспечения, работая в *Intuit* над созданием QuickBooks, наблюдал ту же проблему. «Первая версия QuickBooks должна была выйти спустя девять месяцев. Мы правильно решили, что период разработки будет длиться столько же, сколько беременность, правда, кое в чем слегка ошиблись: срок подготовки проекта составил почти два с половиной года – как беременность слона».

Скотт Макгрегор, архитектор программного обеспечения, отмечает, что в такой ситуации применим также закон Грешема, гласящий: «Худшие деньги вытесняют из обращения лучшие». Если на рынке присутствует две валюты, то более сильную люди хранят, а тратят более слабую. В результате это выливается в то, что в обращении преимущественно остается слабая валюта. Аналогичным образом недостоверная оценка сроков реализации проекта вытесняет реалистичные прогнозы. Если все только и занимаются тем, что делают слишком оптимистичные прогнозы, взятые «с потолка», то руководитель, который пытается предлагать более долгие, но вместе с тем приближенные к реальности сроки, выглядит так, будто умышленно саботирует процесс разработки. В конечном счете на него будет оказано давление, и ему придется скорректировать свою оценку ситуации в сторону уменьшения времени работы над проектом.

Невыполнимость дедлайнов в некоторых проектах вызвана тем, что сроки устанавливают совершенно произвольно. Большинство рационально мыслящих руководителей склоняются к установке таких сроков, которые в целом кажутся достижимыми, однако ради этого приходится идти на огромные жертвы. Это все равно, как если бы пилот самолета вдруг заявил: «Прибудем в Чикаго точно вовремя, но только если сбросим весь багаж!» В своей практике я наблюдал, как руководители проектов по разработке помимо проектирования жертвуют также тестированием, функциональностью, опциями, интеграцией, документацией и даже сущностью проекта. Значительная часть из тех, кто руководит разработкой продукта и с кем мне доводилось работать, предпочтут поскорее вывести на рынок провальный продукт, не желая рисковать оказаться среди опоздавших.

Продукт, который не выйдет никогда

Руководители разработки программного обеспечения нередко предпочитают именно такой подход, потому что глубоко внутри них присутствует страх, что стоит им опоздать с выпуском продукта, и он не выйдет уже никогда. Истории о продуктах, которым так и не суждено было увидеть свет, отнюдь не безосновательны. Сначала выпуск продукта запаздывает на год, потом год превращается в два, а на третий год его настигает карающая десница высшего менеджмента или совет директоров и окончательно умерщвляет. Оттого и возникает неистовая приверженность к дедлайнам, даже если это влечет угрозу жизнеспособности продукта.

Так, в конце 1990-х годов один громко разрекламированный стартап *Worlds, Inc*., команда которого состояла из множества умных и способных специалистов, работал над созданием виртуального онлайн-мира, где аватары людей могли бы перемещаться по виртуальному пространству и вовлекать других аватаров в общение в реальном времени. У продукта никогда не было полного определения или описания, и после того, как на этот проект были спущены десятки миллионов долларов инвестиций, руководство стартапа благоразумно свернуло эту затею.

В начале 1990-х годов еще один стартап *Nomadic Computing* потратил почти 15 миллионов долларов на попытку создать новый продукт для бизнесменов, обладающих высокой степенью мобильности. К несчастью, ни один человек в компании не мог точно сказать, что это за продукт. Команда понимала, на какой рыночный сегмент она ориентируется и каким функционалом должен обладать продукт, но не имела ясной картины о целях потенциальных пользователей. Как безумный скульптор откалывает кусок за куском от мраморной глыбы, надеясь обнаружить внутри статую, так и программисты писали внушительные объемы кода, по сути своей бесполезного, который затем просто выкидывали вместе с деньгами, временем, репутацией и

своей карьерой. Самой же обескураживающей потерей была упущенная возможность сделать по-настоящему желанный для пользователя продукт.

Не удалось избежать такой напасти даже корпорации *Microsoft*. Впервые попытавшись создать программное обеспечение для управления базами данных в конце 1980-х годов, компания затратила на это множество человеко-лет, и завершился этот проект тем, что Билл Гейтс просто-напросто милосердно его закрыл. Ударной волной прошла эта преждевременная кончина проекта по сообществу разработчиков. Последовавший за ним программный продукт Access стал совершенно новой попыткой, над которой работала другая команда разработчиков и другие руководители.

Поздний выпуск продукта – это не смертельно

Как ни странно, поздний выпуск продукта чаще всего не является таким уж фатальным для него событием. Если опоздавший продукт обладает качеством ниже среднего, то он, безусловно, имеет все шансы провалиться, но если продукт несет ценность потребителю, нарушение графика выпуска с большой долей вероятности не вызовет продолжительный негативный эффект. Для настоящего хита не важно, вышел он на месяц или даже на год позже. Microsoft Access вышел с опозданием на несколько лет, тем не менее он все еще занимает значительную долю рынка. И обратная ситуация – окажись продукт отвратительным, кто вспомнит, что он был выпущен точно в срок?

Разумеется, для некоторых массовых продуктов сроки могут быть очень критичны, например, если такие продукты приносят основную долю прибыли только в сезон рождественских праздников. Однако большинство программных продуктов, даже потребительских, не столь требовательны к срокам.

Так, компьютеру PenPoint, разработанному компанией *GO* в 1990 году, пророчили славу прародителя поколения носимых карманных компьютеров и основоположника революции в этом направлении. Однако в 1992 году PenPoint потерпел крах, и эстафету по перевороту индустрии носимых устройств перехватил компьютер Apple Newton. После того как и Newton не смог воодушевить пользователей, на горизонте эры карманных компьютеров забрезжила новая надежда – Magic Link от *General Magic*. На календаре был 1994 год. Когда продажи Magic Link также провалились, на рынке карманных компьютеров наступила мертвая тишина. Инвесторы провозгласили этот сегмент рынка бесперспективным. А потом, словно из ниоткуда, в 1996 году на рынке возник PalmPilot и в мгновение ока снискал всеобщее признание. Его создатели вступили на никем не захваченную территорию *с опозданием на шесть лет*. Рынок всегда готов принять качественный продукт, доносящий ценность и удовлетворяющий потребности клиентов.

Нет причин спорить, что компании, долгие годы создававшие только аппаратные устройства, сегодня прибегают к гибридной технологии, объединяя микрочипы и программное обеспечение. При этом они часто недооценивают важность программного обеспечения и пытаются подчинить процесс его разработки уже сложившимся процедурам создания аппаратного обеспечения. И такая позиция в корне неверна, ведь, как уже было описано в главе 1 «Загадки информационной эры», такие компании теперь ведут свою деятельность в сфере разработки программного обеспечения, знают они об этом или нет.

Выторговывание опций

Следствием процесса управления дедлайнами стал феномен, который я называю «выторговывание опций». Много лет назад программистов обвиняли во всех грехах из-за их привычки запутанно описывать свойства программного продукта на салфетках во время вечеринок, что, по мнению пользователей, нередко влекло за собой появление неудачного ПО. Пытаясь защититься, программисты начали выдвигать требования, чтобы руководители и маркетологи были более точными в своих запросах. Компьютерные программы работают на основе процедур, а эти процедуры с легкостью приводят к появлению новых опций продукта, так что вполне естественно, что программисты соотнесли понятие точности в запросах со списком опций. Этот список опций позволял программистам сваливать всю вину на руководителей, если продукт оказывался провальным. У них всегда находился такой ответ: «Мы здесь ни при чем. Мы добавили все опции, которых требовало от нас руководство».

Как следствие, жизненный цикл большинства продуктов начинается с документа, который имеет разные названия: «маркетинговая спецификация», «техническая спецификация» или

«маркетинговые требования». Другими словами, этот документ представляет собой список желаемых опций, что-то вроде списка ингредиентов в рецепте торта. Как правило, такой документ рождается в ходе множества продолжительных мозговых штурмов, на которых руководство, специалисты по маркетингу и программисты выдумывают, какие опции сразят всех наповал, и делают краткие тезисные наброски. Излюбленным инструментом для создания списков опций обычно являются электронные таблицы, одна такая таблица может быть длиной в десяток экранов. (Одна из строк этой таблицы неизменно содержит опцию «хороший пользовательский интерфейс».) Предложения по новым опциям могут также исходить от фокус-групп, появляться в ходе исследований рынка и анализа конкурентов.

Затем этот сформированный список опций руководители передают программистам со словами: «Продукт должен быть готов к 1 июня». Программисты, конечно же, соглашаются, но с некоторыми оговорками. Они заявляют, что функций слишком много, их не получится реализовать в отведенные сроки, а потому многие возможности придется урезать, чтобы успеть уложиться в дедлайн. Так начинается старый как мир процесс торга.

Программисты проводят черту в середине списка. Все, что выше этой черты, будет реализовано, говорят они, а все, что ниже «линии смерти», откладывается на потом или вовсе игнорируется. Они ставят руководство перед выбором: дать больше времени на разработку или урезать количество опций. Несмотря на то что на разработку продукта неизбежно потребуется больше времени, чем было запланировано, руководство не хочет разыгрывать этот козырь уже в самом начале проекта, потому начинается обсуждение функциональных возможностей. На этом этапе возникает множество споров и моментов, не лишенных драматизма. Опции выменяваются на время, а время падает жертвой опций. Эта примитивная словесная борьба за выгоды так естественна для природы человека, что ни одна из сторон не испытывает неудобств. Придумываются сложные параллельные стратегии. Как отмечает Ройял Фаррос из компании *T/Maker*: «Стоит обвинить какую-нибудь одну опцию в срыве дедлайна, как в список незаметно прокрадется еще с десяток других запоздалых опций». В этой битве теряется видение перспективы, столь необходимое для успешности продукта.

По рассказу Фарроса, флагманским продуктом компании *T/Maker* был текстовый процессор WriteNow – «идеальный программный продукт для университетов. В 1987 году количество продаж копий WriteNow университетам было больше, чем количество продаж копий Word компанией *Microsoft*. Тем не менее мы не смогли удержать лидерские позиции, потому что не учли одной функции, крайне необходимой текстовым процессорам для университетов, – *концевые сноски*, чем сильно разозлили наших самых лояльных потребителей в этом рыночном сегменте. В попытках не сорвать дедлайн мы так и не смогли включить ее в спецификацию. В результате продукт вышел точно в срок, но при этом мы потеряли целый сегмент рынка».

Программисты правят бал

Программисты всецело держат в своих руках контроль над процессом принятия решений, которые на самом деле исходят снизу вверх, хотя кажется наоборот. Именно от них зависит, как долго будет реализовываться каждая опция, поэтому они могут настоять на том, чтобы переместить какой-либо пункт списка в конец, просто заявив, что на его создание уйдет много времени. При этом, если программисты встречают размытое требование, они, в целях самозащиты, относят его к трудоемким, и часто такими требованиями оказываются весьма существенные аспекты пользовательского взаимодействия. В итоге эти аспекты оказываются внизу списка, в то время как наверх выплывают более понятные формулировки и простые в разработке элементы: меню, мастеры создания и установки, диалоговые окна. Все эти аспекты, появившиеся в ходе долгих часов анализа и тщательного обдумывания властями предержащими высокооплачиваемыми топ-менеджерами, в одночасье ставятся под сомнение односторонним решением программиста, действующего из собственных соображений или стремящегося отстоять свою территорию.

Положение руководителей в этом случае становится весьма незавидным – они могут оказывать влияние только на самые незначительные аспекты процесса разработки. Это все равно что пытаться прибавить громкость колонок, которые находятся далеко за пределами

слышимости. Разумеется, контроль процесса создания и выпуска успешных программных продуктов – это необходимость для высшего менеджмента, однако, к сожалению, такое поклонение дедлайнам пренебрегает аспектами, способными привести продукт к успеху, и фокусируется на самом факте создания продукта. Мы отдаем бразды правления в руки разработчиков продукта, тем самым оставляя для руководителей лишь роль пассажиров или пассивных наблюдателей.

Опции – не всегда залог успеха

На самом деле пользователи не слишком нуждаются в большом количестве опций, хотя часто кажется, что это не так. Такое утверждение неоднократно доказывают истории взлетов и падений различных программных продуктов. Пользователям важно лишь то, насколько удобно решать нужные им задачи с помощью конкретного продукта. В редких случаях опции тоже могут пригодиться при решении задач, но чаще случается так, что опции только затрудняют работу и запутывают пользователя. Более того, опции, польза которых неочевидна, заставляют пользователей чувствовать себя глупыми. Если рассмотреть предыдущий пример с компьютером PalmPilot, мы увидим, что этот продукт предлагал гораздо меньше опций, чем провальные Magic Link от *General Magic*, Newton от *Apple* и компьютер PenPoint. Продукт PalmPilot стал успешным благодаря участию в разработке проектировщиков, которые полностью сосредоточились на потребностях целевой аудитории.

То хорошее, что я могу сказать об опциях, заключается в том, что их можно измерить количественно. Это свойство формирует вокруг них мнимый ореол ценности, хотя на самом деле никакой ценности в них нет. Их недостатки перевешивают все достоинства. Самая большая проблема дизайна и проектирования состоит в том, что каждая упомянутая опция, которая может оказаться полезной лишь *потенциально*, заслоняет собой небольшое количество *по-настоящему* полезных опций. Разумеется, реализация таких опций требует определенных затрат. С добавлением каждой из них продукт еще больше усложняется. Они тянут за собой увеличение размера и сложности документации, а также сетевой справочной системы. В дополнение к перечисленному они требуют затрат на увеличение штата специально обученных сотрудников технической поддержки для консультирования пользователей по поводу этих опций.

В мире, помешанном на опциях, может показаться неожиданной мысль о том, что элементарно невозможно достичь каких-либо целей, используя набор опций в качестве инструмента решения задач. Даже если все опции из списка будут успешно реализованы, продукт в конечном счете все равно может оказаться провальным. Проектировщик взаимодействий Скотт Макгрегор в целях доказательства правомерности этого утверждения использует на своих занятиях восхитительный тест, который заключается в следующем: он просит слушателей угадать продукт по описанию списка опций и немедленно записать свои догадки. Затем он начинает перечислять: 1) двигатель внутреннего сгорания; 2) четыре колеса с резиновыми шинами; 3) трансмиссия, связующая двигатель и ведущие колеса; 4) двигатель и трансмиссия, установленные на металлическом шасси; 5) рулевое колесо. К этому моменту каждый слушатель уже с полной уверенностью записал, что продукт является автомобилем, но Скотт продолжает, заканчивая описывать опции, вместо этого упоминая две потребности пользователя: 6) позволяет быстро и легко срезать траву; 7) позволяет сидеть на нем с комфортом. Только лишь на основании описания пяти опций никто не может точно определить, что искомый продукт – это трактор-газонокосилка. Вы сами могли убедиться, насколько более наглядными являются потребности и цели пользователя, нежели опции.

Метод итераций и миф о непредсказуемом рынке

В индустрии, где так много денег и способов их заработать, нередко представляется более простым решение вложиться в новое предприятие, списывая предыдущие провалы на случайность, чем приписать их какой-либо *реальной* причине.

В начале 1990-х годов мне довелось оказаться участником одного из таких провалов. Я помогал создавать компанию, запуск которой осуществлялся на деньги инвесторов, а основной ее целью было заявлено следующее: максимально упростить процесс объединения компьютеров в сеть[8]. Продукт отлично функционировал и был легок в использовании, но ряд грубых маркетинговых ошибок сработал против него и, как это ни прискорбно, привел его к провалу.

Недавно на одной из конференций я столкнулся с инвестором и бывшим членом совета директоров той злосчастной компании. Мы не виделись с тех самых пор, как проект провалился, и, подобно ветеранам, вместе прошедшим через поражение на поле боя и встретившимся спустя много лет, мы утешили друг друга тем, что получили печальный опыт, но зато стали мудрее. Однако, к моему неприкрытому удивлению, этот в других отношениях невероятно успешный и рассудительный человек вдруг заявил, что, окидывая взглядом прошлое, он вынес из него главный урок: несмотря на безупречный маркетинг, менеджмент и техническую реализацию проекта, покупатели «просто не были заинтересованы в легком создании локальных сетей». Я был парализован от его столь очевидно безумного заявления и попытался возразить, что, разумеется, дело было вовсе не в отсутствии потребности, а в том, что мы не смогли эту потребность должным образом удовлетворить. Он переформулировал свое высказывание, настойчиво убеждая меня в том, что мы наглядно доказали отсутствие у покупателей потребности в упрощенном создании сетей.

Позже в тот вечер, когда я рассказывал про эту встречу супруге, я вдруг понял, что оправдание провала, предложенное моим бывшим коллегой, весьма удобно для всех участников этой истории. Обвинив в неудаче непредсказуемый и непостоянный рынок, мой коллега таким образом реабилитировал инвесторов, руководство, маркетологов и разработчиков. И реальность состоит в том, что каждый, кто участвовал в том проекте, в итоге нашел себе место в другом успешном стартапе Кремниевой долины. Инвестиционный портфель этого венчурного инвестора пополнился другими успешными компаниями.

Когда мы работали над нашим продуктом по объединению компьютеров в сеть, мы описывали все его функции в виде списка. Проект не вышел за рамки бюджета. Вывод продукта состоялся в указанные сроки (на деле мы неоднократно переносили выпуск, но в конечном счете продукт был выпущен по графику). Все, что можно было измерить количественно, находилось в пределах нормы. Единственное, на чем могло основываться предположение умудренного опытом инвестора, – это что рынок подвергся возникновению непредвиденной аномалии. Как мог продукт провалиться, если все наши «индикаторы» свидетельствовали о том, что ситуация в норме?

Тот факт, что подобные измерения объективны, вселяет уверенность в участников процесса. Программисты и предприниматели питают невероятное уважение к таким объективным количественным показателям. За всем этим упускается из виду неэффективность подобных измерений применительно к потенциальному успеху продукта. Если продукт становится успешным, основатели считают это личной заслугой, восхваляя свое великолепное понимание потребностей рынка и возможностей технологии.

И напротив, если продукт терпит крах, никто не желает копаться в его останках и выявлять причины неудачи. Любое оправдание будет на руку игрокам, окажись у руководства и разработчиков возможность перекинуться на реализацию нового высокотехнологичного проекта, которых вокруг превеликое множество. Таким образом, повода расстраиваться из-за неудачи нет. Однако неприятная особенность нежелания разбираться в причинах неудач заключается в том, что все участники молча признают невозможность спрогнозировать успех, считая, что в индустрии высоких технологий все зависит лишь от удачи и случая. Это явление, в свою очередь, положило начало подходу к инвестированию, которое инвесторы называют spray-and-pray («стреляй куда придется и молись, чтобы попало»): небольшие суммы денежных средств вкладываются во множество предприятий, а затем остается лишь надеяться на то, что хотя бы одно из них окажется успешным.

* * *

Метод небольших итераций также характерен для сред быстрой разработки, к которым, например, относится Всемирная паутина, а до ее появления таким был Visual Basic. Согласно этому методу, нужно повторять итерации до тех пор, пока какая-то из них не приведет к успеху. Всемирная паутина стала новым средством рекламного продвижения и тем самым привлекла множество специалистов по маркетингу, которые невероятно восприимчивы к мифу о непредсказуемом рынке и вытекающей из этого необходимости к итерациям. Мир рекламы и медиа суров и своеволен, и маркетологи знают это не понаслышке. В конечном счете

значительная часть рекламных кампаний – это *в самом деле* лишь случайные предположения. Например, в рекламном продвижении одним из самых эффективных приемов считается представление какого-либо продукта как «новинки», тем не менее, когда компания *Coca-Cola* в середине 1980-х годов вывела на рынок свою новинку New Coke, она потерпела поражение. Такой исход никто не мог спрогнозировать. Вкусовые предпочтения и поведение людей способны измениться самым непредсказуемым образом, оттого может казаться, что эффективность маркетинга также зависит от случая.

Если говорить о Всемирной паутине, то в ней подобная проблема наблюдается при трансформации веб-сайта из онлайн-каталога в полноценный интернет-магазин. Из одностороннего представления данных он превращается в интерактивный программный продукт. Специалисты из сферы рекламы и медиа, преуспевшие во взаимодействии с простыми веб-сайтами первого поколения, пытаются поступать тем же образом с интерактивными сайтами, применяя к ним метод итераций, что в итоге подводит их к неудаче, хотя они этого даже не осознают. Маркетинг может привести к случайному результату, а вот процесс взаимодействия – нет. Когнитивное сопротивление, появившееся вследствие перехода к интерактивному программному обеспечению, – вот что порождает ощущение неопределенности у неопытного пользователя.

Отчасти этому способствует невероятно гибкое устройство Всемирной сети, благодаря которому стоимость рекламных и маркетинговых кампаний, равно как и количество времени, которое нужно на это затратить, в разы меньше, чем в случае с печатной и телевизионной рекламой. Находчивому интернет-маркетологу доступна практически молниеносная обратная связь об эффективности рекламы, поэтому скорость прохождения итераций возрастает кратно, в связи с чем отдельные изменения в продукт можно вносить в течение одного дня. На практике же это выливается в ситуацию «рискнем сделать так, авось что-то из этого выйдет». Многие руководители интернет-стартапов поступают именно так, придерживаясь этого поразительно простого подхода к проектированию методом случайных предположений. Они клонируют какую-нибудь старую программу, затрачивая на это минимум времени, а затем предлагают пользователям то, что получилось. После этого они собирают обратную связь и жалобы пользователей, изучают карты кликов, дорабатывают слабые места программы и выпускают следующий релиз.

Обычно программисты не слишком увлечены разработкой по методу итераций, потому что объем их работы резко увеличивается. Как правило, именно руководители, для кого технология является новой, становятся ярыми поклонниками итеративного процесса разработки, ведь таким образом им не приходится тщательно планировать, много думать и проводить комплексную оценку рисков выпуска продукта (другими словами, они освобождаются от проектирования взаимодействия). Разумеется, от этого больше всего страдают пользователи. Им приходится мучительно продираться сквозь жалкие программные недоделки, раз за разом, до тех пор пока они наконец не получат программу, которая не будет причинять им столько страданий.

Несмотря на то что обратная связь от потребителей улучшает ваше понимание продукта или услуги, нельзя руководствоваться только этими соображениями и уж тем более нельзя считать эффективным, дешевым или действенным такой подход, при котором на пользователя вываливается случайный набор опций, а затем отслеживается, какие ему понравятся, а какие нет. В мире медведей-плясунов такая стратегия может обладать некоторой долей жизнеспособности, но на любом другом рынке, где есть хоть малейший намек на конкуренцию, такой подход приведет к краху. Даже в случае, когда на рынке кроме вас никого нет, эта стратегия выльется в большие затраты.

Многие в других отношениях разумные и высокопрофессиональные руководители отчего-то беззастенчиво гордятся таким подходом. От одного зрелого и опытного исполнительного директора (в прошлом работавшего в сфере маркетинга) я как-то услышал весьма самонадеянный вопрос: «Как может кто-либо осмеливаться утверждать, что знает, чего хотят пользователи?» Это просто поразительно. Каждый предприниматель осмеливается это утверждать. Тем и ценны многие руководители, что привносят в свою сферу подобные «предположения» о том, чего хотят потребители. Разумеется, в отношении *некоторых* пользователей эти предположения окажутся неверными, но полное отсутствие предположений приведет к тому, что продукт не оценит *никто*. Этот глупец полагал,

что пользователи не станут возражать против того, чтобы с трудом продираться сквозь «непаханое поле» программы безо всяких предположений, таким образом выполняя за него его работу по проектированию. Сегодня в Кремниевой долине наверняка нашлось бы множество любящих гулять по сети апологетов, кто хотел бы помочь этому ленивому руководителю постичь сущность его бизнеса, но при этом скольких страдающих потерпевших он отвратил своим дурным отношением? Пока он выпускал все новые и новые зачатки своего сайта, реагируя лишь на тех, чья выдержка позволила им снова вернуться на сайт, скольких клиентов он потерял безвозвратно? Что было нужно *им*? Говорят, Сталин использовал для разминирования минных полей солдат-пехотинцев, которым приказывал по ним проходить. Эффективен ли такой способ? Да. Но можно ли считать его рациональным, гуманным, целесообразным, желанным? Нет.

Разумеется, самым серьезным недостатком этого подхода можно считать то, что потерпевшие сразу отвергают его использование и остаются только апологеты. Такая ситуация существенным образом сказывается на сущности и качестве обратной связи, обрекая разработчиков на взаимодействие только с технически подкованными апологетами, что составляет весьма небольшой сегмент рынка. Это одна из причин, по которой так мало компаний, занимающихся разработкой программных продуктов для персональных компьютеров, достигают успеха на массовых рынках.

Я совсем не имею в виду, что не нужно учиться методом проб и ошибок, но эти пробы должны иметь под собой более весомое основание, чем случайность, и должны начинаться с тщательно взвешенного решения, а не с того, что вдруг в голову взбрело. В противном случае это просто даст ленивым и безразличным предпринимателям официальное разрешение злоупотреблять благосклонностью потребителей.

Скрытые издержки плохого программного обеспечения

Люди склонны избегать работать со сложными и раздражающими их программами. Поначалу этот факт не кажется таким уж важным, пока не приходит осознание, как много людей зависит от программного обеспечения при решении рабочих задач. Затраты компаний на такое избегаемое программное обеспечение невозможно измерить, однако они вполне реальны. Обычно эти затраты не имеют денежного выражения, а проявляются в виде более дорогой валюты – времени, порядка, репутации и потребительской лояльности.

Те, кто пользуется программами для решения бизнес-задач, могут относиться к ним с презрением, но они получают плату за свою терпимость. Как следствие, отношение людей к программам постепенно изменяется. Ввиду того, что им платят за использование этого программного обеспечения, люди готовы с большей терпеливостью относиться к его недостаткам, ведь у них нет выбора поступить иначе, только использование такого ПО не становится от этого менее затратным. Совсем наоборот – затраты по-прежнему высоки, но вот выявить и учесть их уже не так просто.

Из-за плохо спроектированного программного обеспечения для решения рабочих задач пользователи начинают ненавидеть свою работу. Производительность труда падает, работа

начинает выполняться с ошибками, пользователи пытаются саботировать программы, а в итоге и вовсе увольняются. Потеря сотрудника обходится компании очень дорого, и дело даже не в финансовых потерях, а в том, что в работе возникают простои: а потерянное время вернуть невозможно. Несмотря на то, что многие пользователи стыдятся открыто выражать недовольство программными инструментами, ведь работа им оплачивается, ничто не мешает им чувствовать раздражение и неприязнь к программам.

Одна из самых больших затрат, которые несет компания из-за сложных в использовании программ, – это необходимость технической поддержки пользователей. Затраты *Microsoft* по этой статье расходов составляют 800 миллионов долларов в год. И это несмотря на то, что *Microsoft*тратит многие сотни миллионов долларов на исследования и юзабилити-тестирование. Нет сомнений в твердой убежденности руководства компании в том, что такие масштабы технической поддержки являются неизбежными издержками. Однако у меня другое мнение на этот счет. Только вообразите, насколько более эффективной станет ваша компания, если вы будете думать иначе, чем это делает *Microsoft*. И насколько большие перспективы в разработке перед вами откроются, если вы не будете затрачивать свыше пяти процентов дохода на содержание службы технической поддержки.

Задайте вопрос любому из тех, кому довелось поработать в технической поддержке любой из компаний, занятой в сфере создания настольных программ, и он непременно ответит, что больше всего времени и сил он тратит на объяснение нюансов работы с файловой системой. Пользователи, так же как Джейн из главы 1, не разбираются в рекурсивной иерархии файловой системы, и не важно, будет ли это Finder или Explorer в операционной системе Windows, Mac или UNIX. Удивительно, но очень невелика доля тех компаний, которые тратят средства на проектирование и реализацию версий файловых систем, более дружественных пользователю. В большинстве своем компании предпочитают более дорогой путь в виде содержания службы технической поддержки для ответов на бесконечные вопросы пользователей.

Можно сколько угодно обвинять «чайников» во всех грехах, но вам в любом случае придется нести значительные затраты на высокооплачиваемых телефонных консультантов технической поддержки, если ваша компания планирует продавать плохо спроектированное программное обеспечение.

Дороже разработки программ может быть только разработка плохих программ

Каждый программист стоит немалых денег, потому программисты, праздно ожидающие окончания этапа проектирования, невероятно раздражают руководителей. Ведь это же бессмыслица – программисты просто сидят и ждут, хотя могли бы в это время писать код, – рассуждает руководитель. Как бы то ни было, вынуждать программистов работать до того, как проектирование будет завершено, – это мнимая экономия. Как только процесс написания кода запущен, его уже не остановить, и в этом случае процесс проектирования приходится подстраивать под нужды программистов, хотя должно быть ровно наоборот. Заставлять программистов ждать – и вправду решение не слишком благоразумное, однако здесь можно применить одну маленькую несложную хитрость. Поручите проектировщикам взаимодействия планировать следующий продукт или релиз параллельно с этапом написания кода для текущего продукта или релиза, и тогда ваши программисты никогда не будут сидеть сложа руки.

В долгосрочной перспективе гораздо более затратным мероприятием может оказаться написание ненужного кода, чем вообще всякое отсутствие кода. Большинству руководителей эта мысль кажется настолько парадоксальной, что они просто не принимают ее всерьез. После того как код уже написан, весьма затруднительно от него избавиться. Как писатели нежно любят свои произведения, так и программисты испытывают некую эмоциональную привязанность к своим алгоритмам. Внесение непредвиденных изменений в программу посреди процесса разработки не только способствует разладу, но и вредит самому коду. Руководителю избавиться от лишнего кода еще труднее, ведь он знает, во сколько обошлось его создание, и прекрасно понимает, что замена выльется в еще большие траты.

Если же этап проектирования не выполняется прежде этапа программирования, то никакого толку от него не будет. Как-то от одного руководителя я услышал такую фразу: «Мы уже поручили нашим сотрудникам писать код, и я не собираюсь останавливать их». Посыл этого умника приблизительно такой: «Я успею соорудить парашют прежде, чем мы ударимся о землю». Звучит весьма смело, но я ни разу не видел, чтобы это сработало.

Без серьезной основы в виде плана проектирования программисты начинают бесконечные эксперименты со своими программами, пытаясь найти оптимальное решение. Как в случае с плотником, «на глаз» отпиливающим куски доски, пока не получится подходящий для латания дыры в стене, этот метод влечет избыточные потери.

Из-за того, что программное обеспечение не поддается количественному измерению и кажется неосязаемым, становится практически невозможно оценить его масштабы и степень завершенности. Прибавьте к этому пристрастие программистов к своей работе, и вам станет понятно, что разработки только ширятся в объеме и времени и никогда не становятся меньше. Если мы продолжим практиковать такой подход к программированию, нас всегда будут поджидать неожиданности, до тех пор пока мы не станем назначать промежуточные контрольные точки и достоверно отмечать степень нашего прогресса по отношению к ним.

Цена упущенных возможностей

С приходом информационной эры создать что-либо становится менее затратно, чем упустить возможность создания. Провал продукта означает, что вам не удалось создать успешный продукт. Если в течение трех лет вы выпускали по одной версии продукта в год, надеясь в итоге создать успешный, это означает, что в течение трех лет вы три раза не создали хороший продукт.

Основной деятельностью компании *Novell* было создание программных продуктов для управления сетями, однако эта же компания пыталась открыто тягаться с *Microsoft*, выводя на рынок офисные приложения. Эти попытки стоили *Novell* очень дорого, но самой большой потерей для компании стала утрата лидерских позиций на рынке сетевого ПО. Денежные потери ничто в сравнении с упущенными возможностями.

Похожая история произошла и с компанией *Netscape* – она потеряла лидерство на рынке браузеров в тот момент, когда решила попытаться обойти *Microsoft* в сегменте операционных систем.

Каждый, кто разрабатывает продукты, в основе которых лежат кремниевые микросхемы, обязан сосредоточиться на оценке и изучении главных целей потенциальных потребителей и сфокусироваться на том, чтобы помочь им этих целей достичь. Ведь так легко упустить большую возможность в сфере высоких технологий, распыляясь на заманчивое множество более мелких. Какими бы умными, проницательными, преданными ни были программисты и какими бы благородными намерениями ни руководствовались, все же ими движут несколько иные побуждения, а потому они способны с легкостью увести компанию в сторону от ее основных целей.

Издержки прототипирования

Прототипирование представляет собой фактически то же самое, что и программирование – движущая сила и размер затрат у них схожи, разница в том, что результат прототипирования не столь гибкий, как качественный программный код. Прототип программного обеспечения сравним со строительными лесами – он имеет мало общего с настоящими каменными стенами, каковыми является долговременный, управляемый, расширяемый программный код. Тем не менее руководители, как правило, неохотно расстаются с работающим кодом, даже если тот используется для прототипа. Они не могут четко отделить строительные леса от каменных стен.

Создать прототип можно в разы быстрее, чем настоящую программу. Потому он выглядит таким привлекательным, ведь этот процесс кажется незатратным, однако результатом программирования является надежная программа, а результатом прототипирования – лишь неустойчивая основа. Прототип по сути своей – только эксперимент, и от его результатов нужно вовремя избавляться, хотя на практике мало кто поступает подобным образом. Руководители видят рабочий прототип и интересуются: «А почему бы нам просто не воспользоваться этим?» Возможный ответ покажется переполненным техническими нюансами и неясными моментами,

поэтому переубедить руководителя будет достаточно сложно, учитывая, что он видит в этом способ избежать многомесячных дорогостоящих усилий.

Вся суть хорошего программирования заключается в том, что вознаграждение придет позже. Сначала вы выкладываетесь на максимум, а уже потом пожинаете плоды своих трудов. В мире не так много задач, выполнение которых вручную будет стоить больше. Тем не менее однажды написанная программа может быть выполнена миллион раз без дополнительных затрат. Самой дорогой будет программа, которую запустят лишь раз. Самой дешевой – та, которая отработает десять миллиардов раз. Если абстрагироваться от несерьезных программ, вроде тех, что пишут в школьные годы, то можно увидеть, что экономика программного обеспечения претерпела странные изменения: самыми дешевыми программами являются те, что влекут больше всего затрат при разработке, а самыми дорогими – те, написание которых обходится в разы дешевле.

Написание большой серьезной программы похоже на выкладывание башни из кирпичей. Башня представляет собой тысячу кирпичей, уложенных один на другой. Построить такую башню можно, только если выкладывать кирпичи очень аккуратно. Любая небрежность может повлечь крен и разрушение башни. При этом, если неровно положить 998-й кирпич, то башня, вероятно, достигнет высоты в тысячу кирпичей, а вот если сдвинуть пятый кирпич, то она едва ли доберется до двадцать пятого.

То же самое характерно и для программного обеспечения – манипуляции с его фундаментом могут повлечь гораздо бóльшие неприятности, нежели изменения кода более высокого уровня.

В процессе создания любой программы разработчикам свойственно допускать промахи в самом начале и вносить изменения по ходу действий. В результате программный код покрывается «рубцами» от внесения многочисленных изменений. В любой программе можно найти функции, которые не используются, и обрывки недоделанных возможностей. В любой программе также есть опции и инструменты, необходимость которых возникла уже после начала разработки, и они были «прилеплены» к коду позже. Каждый из этих рубцов вносит свой маленький вклад в отклонение башни от вертикали.

Перенести кнопку от одного края диалогового окна к другому – это все равно что слегка сдвинуть кирпич под номером 998, а вот внесение изменений в код, отвечающий за отображение всех кнопок, – это сдвиг пятого кирпича. В этом случае объектно-ориентированное программирование и инкапсуляция данных – это защитные меры, единственная цель которых – уберечь программу от появления подобных рубцов. Фактически объектно-ориентированный подход превращает башню высотой в 1000 кирпичей в десять башен по 100 кирпичей.

Хорошие программисты тратят невероятно много времени и сил на подготовку к написанию серьезной программы. На то, чтобы настроить среду программирования, прежде чем написать хотя бы одну строку кода, может уйти много дней. Нужно произвести тщательный отбор подходящих программных библиотек. Определить массив данных, с которыми будет производиться работа. Провести анализ и определить возможности подсистем хранения и возврата данных, описать их в коде и протестировать.

Чем дальше программисты погружаются в работу, тем лучше они видят упущения в планировании и промахи в первоначальных предположениях. Им предстоит сделать так называемый гобсоновский выбор – потратить дополнительное время и усилия, исправляя все с самого начала, или же наложить локальную «заплатку», оставаясь там же, где они сейчас, тем самым соглашаясь с возникновением нового рубца и – как следствие – отклонения от нормы. Сдавать назад всегда выходит слишком затратно, однако многочисленные рубцы в конечном счете ограничивают размер программы – не дают башне из кирпичей стать выше.

Каждый раз, когда программа модифицируется в целях избавления от багов или расширения функционала, в ней появляются новые рубцы. Именно поэтому старые программы нужно уничтожать и переписывать их с нуля приблизительно каждые двадцать лет. В противном случае программа так сильно обрастет рубцами, что это будет препятствовать ее нормальному функционированию.

Прототипы по сути своей являются программами, слепленными в спешке, чтобы быстро посмотреть, как будет выглядеть результат. Для создания прототипа в сжатые сроки программисту придется пожертвовать аккуратностью в выравнивании кирпичей. Вместо того чтобы использовать «правильные» структуры данных, информацию вбрасывают беспорядочно. Вместо того чтобы использовать «правильные» алгоритмы, в ход пускают первые подвернувшиеся под руку фрагменты кода. Уже в *самом начале* своего жизненного цикла прототип обрастает многочисленными рубцами. Ему никогда не стать большой серьезной программой.

К сожалению, некоторые разработчики пришли к выводу, что современные инструменты для быстрого прототипирования – например, Visual Basic – являются эффективным средством проектирования. Теперь, вместо того чтобы создавать качественный дизайн и проектирование продукта, они просто выдают крайне жалкое подобие программы при помощи инструмента визуального программирования. Этот прототип впоследствии обычно используется как основа для продукта. Жертвой таких призрачных выгод падает надежность продукта и срок его жизни. Вместо создания множества прототипов гораздо лучше удастся спроектировать дизайн продукта, воспользовавшись карандашом, листом бумаги и хорошей методологией.

Конечно, людям, не знакомым с проектированием, бывает достаточно сложно, а порой вообще невозможно визуализировать форму и поведение программы, которая еще не существует. Для этих людей из мира бизнеса прототипы служат инструментом визуализации. Так как прототип является лишь приблизительной моделью, созданной с помощью наиболее подходящих встроенных средств разработки, совершенно естественно, что он содержит множество компромиссных решений. Но программное обеспечение, которое работает, пусть

даже плохо, оказывает впечатляющее воздействие на тех, кто должен оплатить создание программы. Функционирующий, пусть и не всегда гладко, прототип обладает сверхъестественной движущей силой, застилающей его истинную невеликую ценность.

Ах, как же сильно руководителю хочется сказать: «Не избавляйтесь от прототипа. Давайте положим его в основу *реального* продукта». Но такое решение нередко подводит к ситуации, когда продукт так никогда и не выходит на рынок. Программисты оказываются навеки прикованными к необходимости бесконечно возвращать программу к жизни после каждого сбоя, угрожающего ее работоспособности по мере ее развития. Подобно башне, где первые 25 кирпичей положены вкривь и вкось, не имеет значения, насколько ровно вы положите оставшиеся кирпичи, насколько скрупулезно выполняет каменщик свою работу и как крепко держит их строительный раствор – башня падет под силой гравитации где-то на высоте пятидесятого кирпича.

Вся ценность прототипа заключается в том опыте, который вы получаете от его создания, а вовсе не в написанном коде. Вот мудрый совет от разработчика Фредерика Брукса: «Запланируйте избавиться как минимум от одной версии». Вы все равно сделаете это, так что лучше держать это событие под контролем изначально.

В 1988 году Билл Гейтс купил у меня программу под названием Ruby. Это был язык визуального программирования, который впоследствии объединили с продуктом Билла QuickBasic, превратив его в Visual Basic. То, что первоначально увидел Гейтс, было всего лишь прототипом, но он демонстрировал значимые подвижки в применении проектирования и технологии. (Взглянув на это впервые, он воскликнул: «Как тебе *удалось* это сделать?») К работе над проектом Ruby был также привлечен Расс Вернер, в то время руководивший разработкой Windows 3.0. Далее мы заключили сделку, согласно которой я должен был написать полноценную программу и довести проект Ruby до завершения. Первым делом я отбросил прототип и начал все заново, с чистого листа, руководствуясь лишь своим опытом и здравым смыслом. Когда об этом узнал Расс, он был страшно удивлен, зол и разъярен. Никто и никогда еще не поступал таким возмутительным образом, и потому он был убежден, что, отбросив прототип, мы тем самым сорвем сроки выпуска продукта. Но все уже свершилось, и, несмотря на опасения Расса, мы завершили создание программы строго по графику. Интеграция среды VB с языком Basic сделала этот продукт одним из самых удачных ранних релизов для *Microsoft*. В противовес этому событию выпуск Windows 3.0 задержался более чем на год, и с тех пор он пользовался дурной славой неполноценного продукта ввиду чрезмерного количества неиспользуемого кода, оставшегося от прототипа.

В большинстве своем далекие от технологий руководители придают особую ценность именно завершенному коду. Вне зависимости от степени его надежности они предпочитают его дизайн-проекту или даже совету людей, которые этот код писали, и делают это совершенно напрасно. Как-то мой коллега, занимающийся созданием программного обеспечения для автомобильных систем навигации, Клэй Колье, рассказал мне историю об одной системе, над которой работал по заказу крупной японской компании, изготовляющей электронные запчасти для автомобилей. По запросу клиента Клэй разработал прототип пользовательской системы навигации. Как и полагается хорошему прототипу, он подтверждал, что система в целом будет рабочей, но за пределами этого ее функционирование не отличалось надежностью. В один прекрасный день в США прилетел президент этой японской компании, изготовляющей электронные запчасти, и пожелал увидеть программу в действии. Коллега Клэя, назовем его Ральф, понимал, что отказать президенту компании никак невозможно, и решил показать ему демоверсию. Ральф встретил президента в аэропорту Лос-Анджелеса и усадил в автомобиль, который был специально оборудован прототипом навигационной системы. Все, в чем был уверен Ральф, – это что прототип может отобразить путь до офисов компании, но все прочие функции еще не проходили тестирование. К великой досаде Ральфа, президент попросил доставить его в какой-то незнакомый ресторан на ленч. Ральф не знал, где находится этот ресторан, и не мог точно понять, сможет ли прототип отобразить путь. Он скрестил пальцы и ввел в навигатор название ресторана. К его большому удивлению, навигатор начал выдавать инструкции: «Поверните направо в направлении Линкольна», «Перестройтесь в левую полосу» и т. д. Ральф покорно исполнял указания навигатора, в то время как президент безмолвствовал, размышляя о чем-то своем. Так продолжалось до тех пор, пока автомобиль не начал проезжать все более и

более сомнительные районы, – вот тогда Ральф всерьез обеспокоился. В тот момент, когда компьютер велел остановить автомобиль, нервы Ральфа были на пределе. Кто-то подошел к автомобилю снаружи и распахнул дверь. К огромному облегчению Ральфа, это оказался служащий того самого нужного президенту ресторана. На лице президента засияла улыбка.

Как бы то ни было, успешная демонстрация прототипа отразилась на Ральфе отнюдь не самым благоприятным образом. Президент был так впечатлен функционированием «системы», что поручил Ральфу сделать на ее основе готовый продукт. Ральф запротестовал, заявив, что это была только проверка возможности реализации и что она недостаточно надежна, чтобы делать миллионы пользовательских устройств на ее основе. Президент и слышать ничего не хотел. Он своими глазами видел прототип в действии. Ральф уступил, и через восемь долгих лет его компания наконец-то выпустила первый работоспособный релиз продукта. Он работал медленно, содержал множество багов и сильно уступал даже более молодым, новым продуктам конкурентов. Газета *New York Times* окрестила его «полнейшей катастрофой».

Тот опыт и знания, которые Ральф и его команда получили в процессе создания некорректно функционирующего прототипа, были гораздо более ценны, чем непосредственно код. Президент не сумел этого понять и отдал предпочтение коду, отчего пострадала вся компания.

* * *

Если ставить рамки проекта лишь по дедлайнам и набору опций, то даже выпуск точно в срок не сделает его желанным. Если же описывать проект категориями качества и степени удовлетворенности потребителей, то получится востребованный продукт, сроки разработки которого останутся в допустимых пределах. В Кремниевой долине есть одна старая шутка: «Как сколотить небольшое состояние на разработке программного обеспечения? Ответ: разумеется, начать с большого состояния!» Проекты по созданию программного обеспечения, даже под началом опытных руководителей, влекут такие большие скрытые издержки, которые способны поразить даже Дональда Трампа. Если принять во внимание долгосрочную перспективу, то гонки на яхтах и увлечение наркотическими веществами обойдутся дешевле, чем бесконтрольное создание программного обеспечения.

4. Медведь-плясун

Даже в том случае, когда «потерпевшие» явно понимают, что взаимодействие с программным продуктом заставляет их чувствовать себя глупыми, они не могут открыто признать это, ведь такое поведение в среде ярых защитников продукта будет выглядеть просто как жалобы и нытье. Нытиков в обществе не приветствуют, так что «потерпевшие» испытывают сильное социальное давление – их вынуждают перейти в стан апологетов, оправдываться за свое «плохое» поведение и винить во всем самих себя. Но инстинктивно «потерпевшие» догадываются, что правы, несмотря на попытки сознательно убедить себя в некомпетентности. Программы *действительно* заставляют их чувствовать себя глупыми, и это факт, которого быть не должно. Если вы один из таких людей, то, вероятно, сейчас задаетесь вопросом: «А что значит „плохая программа“? Она ведь решает рабочие задачи, разве нет?» В последующей части главы я объясню свое понимание «плохого» в отношении программ.

Почему же эта проблема до сих пор не решена?

Вся проблема программ, похожих на пляшущих медведей, состоит в том, что большинству людей для веселья достаточно и таких неуклюжих танцев. И только когда они видят настоящее танцевальное искусство, в их голову закрадывается подозрение, что мир не ограничивается косолапыми движениями. Среди программных продуктов чрезвычайно мало таких, которые демонстрируют «танцевальное искусство», в связи с чем многие люди находятся в наивном неведении, что все могло бы быть лучше, гораздо лучше. Большинство из тех, кто использует программы для работы с электронными таблицами и текстовые процессоры на своих компьютерах, думают, что все то, что можно было решить с помощью технологий, уже решено, причем адекватно. Однако это далеко не так. Количество задач, связанных с обработкой информации, решение которых не найдено, стремится к бесконечности, при этом значительную часть никто даже и не пытался решить.

Жертва потребительской электроники

Пользуясь различными программными продуктами, мы так привыкли брать все, что нам дают, что уже не помним, чего мы хотели от той или иной программы или устройства изначально. Конечно, нет сомнения в том, что продукты, создаваемые инженерами, подходят для выполнения рабочих задач, однако без надлежащего проектирования этот набор доступных задач все еще недостаточен для достижения целей пользователя.

За двадцать лет я сменил бессчетное количество видеомагнитофонов. Все они обладали функцией отложенной записи передач, но ни с одним из них – даже с самой дорогой топовой моделью за полторы тысячи долларов – я в действительности не был уверен, что у меня получится это сделать. Тот интерфейс, которым они обладали, был настолько сложен в управлении, содержал настолько неудобные обозначения, неясные термины и установки и предлагал так много скрытых опций и режимов, что показатель моей успешности при записи передач приблизительно равнялся 40 %. Больше половины моих попыток оканчивались схожим образом: я обнаруживал, что моя запись содержит трехчасовой бразильский футбольный матч вместо нужной мне передачи с телеканала PBS. Спустя годы сражений с техникой я в итоге признал поражение и оставил всякие попытки записывать телепередачи. Так же поступили и все члены моей семьи. И все мои друзья. Мы – те, кто пострадал от нашествия «медведей-плясунов» под видом программ.

И вот в полном отчаянии я спешу в местный гипермаркет электроники, едва получив банковский перевод на мою зарплатную карту. «Кому тысячу долларов? – кричу я. – Готов дать даже две тому продавцу, который найдет мне нормальный видеомагнитофон для записи телепередач!» Продавцы-консультанты, одетые с иголочки, окружают меня и начинают предлагать свой товар. Выбирай на любой вкус – от самых низкобюджетных до невероятно дорогих вариантов, только в отношении взаимодействия разницы между ними нет. Разумеется, все они наделены бесконечным набором опций, но встроенная программа управления повторяется от модели к модели. Другими словами, продукт «зреет» уже 20 лет, а пользоваться им все так же неудобно. Это и есть «медведь-плясун» в худшем понимании этого эпитета.

Стоит мне указать на это продавцу, как он начинает защищать свой товар, утверждая, что лучшего я в любом случае нигде не найду. В подтверждение своих слов он извлекает брошюру, где говорится, что этот видеомагнитофон «прост в использовании». Однажды Билл Гейтс с нехарактерным для него цинизмом отметил: чтобы сделать программу с дружественным интерфейсом, нужно на каждой упаковочной коробке поставить штамп USER FRIENDLY («Дружественный интерфейс»). Похоже, поставщики компьютеров решили всерьез воспользоваться этим методом.

Кнопки не слишком хорошо подходят для установки значений бесконечных параметров, таких как время, – крутящиеся ручки справились бы лучше. Если бы у этого видеомагнитофона была такая круглая ручка, как у моего одиннадцатидолларового будильника марки Baby Ben, я

спокойно установил бы время и навсегда, как страшный сон, забыл эти мерцающие цифры 12:00. А будь у него еще одна такая ручка, я бы и время записи будущей передачи устанавливал с той же легкостью. На деле же устройство, в котором заложена возможность установить запись для *десяти*передач, непригодно даже для одной записи.

Программы, похожие на пляшущих медведей, повсюду вокруг нас. Их упаковочные коробки снизу доверху испещрены перечислением их функций. У них так много опций, что в сравнительном обзоре программ в популярном журнале они по каждому параметру получат свое «да». Только пользователи не становятся счастливее и производительнее от таких программ. Большинство пользователей не могут извлечь выгоду из всех этих возможностей и опций. На это способны лишь апологеты, охотно готовые изменить собственные привычки в выполнении рутинных задач, чтобы подстроиться к особенностям такого ПО. Они упиваются возможностью закопаться в настройки. Они прилежно постигают все тонкости расширенных функциональных возможностей, которыми едва ли когда-нибудь воспользуются.

Что плохого в почтовых программах

Пока производители программного обеспечения соревнуются в красноречии, стремясь продвинуть свой продукт, пользователи сжимаются за своими рабочими столами, боясь даже подумать, что их еще ждет. Взять, например, разработчиков программ электронной почты – они добавляют в свои продукты все новые и новые функции, забывая, для чего изначально предназначался обмен электронными сообщениями.

Те, кто впервые начинает пользоваться электронной почтой, восторгаются ранее неведомой возможностью коммуницировать с другими напрямую, легко и просто, и к тому же асинхронно. Но просто решить задачу коммуникации не значит помочь пользователю достигнуть его целей, поэтому обмен электронными сообщениями все еще функционирует на самом примитивном уровне. Проблемой является неверное понимание того, для чего применяется электронная почта на самом деле. Двадцать лет назад получение *каждого* электронного сообщения считалось великим событием. Но возвеличил сообщения сам способ их передачи – фактически ничего ценного они собой не представляли. На деле это был просто отдельный файл без каких-либо особых параметров и взаимосвязей, содержащий неформатированные символы из набора ASCII.

В настоящее время всем нам приходит множество как нужных, так и бесполезных электронных писем. Каждый человек, активно использующий электронную почту, быстро понимает, каким мощным и полезным инструментом она является, и начинает применять его все чаще и чаще, пока на него не приходится значительный объем деловых и личных сообщений. Ежедневно на электронные ящики большого количества пользователей поступают десятки и сотни писем. Значительная часть из них приходит в ответ на прошлые сообщения либо в ожидании отклика. Такие связанные сообщения, иначе называемые «цепочками», курсируют туда-обратно между двумя и более собеседниками. Количество цепочек писем на моем компьютере по отношению к одиночным сообщениям составляет приблизительно 50 к 1. И все еще ни один почтовый клиент из доступных на текущий момент не увязывает такие сообщения в последовательность[9]. Все это выглядит так, словно связанных сообщений не существует в принципе или же это несущественный атрибут небольшого количества отдельных писем.

Нетрудно догадаться, что, когда человек просматривает цепочку сообщений, а не каждое в отдельности, он может четко увидеть взаимосвязь между ними, выделить главные темы беседы и отследить, как они сливаются в единый диалог. Если же мы рассмотрим эту проблему с точки зрения функциональной задачи, все, что мы увидим, – это необходимость отправлять сообщения и отвечать на них.

Заложить в программу возможность создавать автоматические цепочки писем – не самая сложная задача для разработчика; все дело лишь в том, что примеров программ, работающих таким образом, еще не было, так что разработчики не готовы блюсти интересы пользователей и внедрять нововведения, а их руководители не готовы идти по непроторенному пути.

Поскольку разработчики рассматривают проблему с точки зрения реализации, они видят входящий и исходящий поток сообщений и то, что эти сообщения пользователи могут распределять по папкам, так что в чем тут проблема, программисты не понимают. Как только им удалось заставить медведя шевелиться, они объявляют это пляской и не предпринимают дальнейших попыток к совершенствованию.

Программа для работы с электронной почтой – не единственный пример программного обеспечения, которое не позволяет эффективно выполнять простые и очевидные базовые задачи. Мы так зачарованы пляшущими медведями, что не замечаем неадекватности всей ситуации. Приведу еще несколько примеров.

Что плохого в программах-планировщиках

В любом бизнесе, связанном с консультированием, будь то юридические, рекламные или бухгалтерские услуги, есть огромная незакрытая потребность в программном обеспечении, которое бы помогло управлять назначением сотрудников на проекты с привязкой ко времени. На этих трех аспектах построена деятельность любой консалтинговой компании, тем не менее, каким бы удивительным это ни казалось, все еще не написана программа, которая бы это учитывала.

С точки зрения разработчика управление проектом сводится к задаче планирования, да еще, возможно, к дополнительным ухищрениям вроде анализа методом критического пути, при котором начало каждой следующей задачи зависит от завершения предыдущей. Все имеющиеся на сегодня программы управления проектами основаны на этом чисто теоретическом предположении[10]. Проблема состоит в том, что такой взгляд на управление проектами имеет мало общего с действительностью.

Одно из базовых предположений программ для управления проектами состоит в том, что пользователям требуется помощь в разъяснении, как им управлять их проектом, – и это в корне неверно. Бóльшая часть пользователей и без того успешно справляется со своими проектами, ведь это, в конце концов, их работа. С чем действительно требуется помощь – это с возможностью одновременно встроить несколько проектов в расписание, так как ресурсы (обычно под этим понимаются люди) могут быть параллельно задействованы в разных проектах. Проекты начинаются и заканчиваются непрерывно, часто возникают накладки, и все прочие проекты временно ставятся в очередь. Поэтому просто распределить ресурсы по проектам недостаточно – в программах должна быть возможность параллельной работы в нескольких проектах сразу.

Чтобы от таких программ управления ресурсами была польза, в них должна осуществляться интеграция трех плоскостей проблемы: времени, проектов, ресурсов. Но мы взамен получаем программы, которые учитывают только две плоскости – ресурсы и время, при этом разработчики таких программ настойчиво убеждают нас, что этих плоскостей нам вполне достаточно. Как бы ни называли подобные программы – «менеджеры трафика», «программы управления проектами», «программы распределения ресурсов», – такой жизненно необходимый сегмент на рынке приложений не представлен.

Помимо этого, проекты склонны изменяться ввиду изменения планов. Любой программе управления проектами, претендующей на полезность, не мешало бы обладать гибкостью и способностью адаптироваться к возникающим обстоятельствам. Система управления проектами без поддержки надежных и продуктивных инструментов обратной связи, которые позволили бы сотрудникам, занятым в проекте, отразить в системе реальное положение дел, не слишком пригодна на практике.

Что плохого в календарях

Практически каждый человек так или иначе пользуется электронным календарем для планирования рабочих задач. Нам доступен выбор из множества программ, но при этом почти все из них упрямо игнорируют самые простые способы, как человек обычно использует календарь. Проще говоря, в календаре должно отражаться то, как люди распределяют время для контроля своей жизни. В целом все объекты, зависимые от времени, можно разделить на две категории: текущие дела и дедлайны. К дедлайнам относят крайние сроки, до которых часть проекта или дела должна быть исполнена; иными словами, это некая контрольная точка хода проектирования. Текущие дела – это, например, командировка на двое суток. А в ходе этой двухдневной поездки в Чикаго у меня, скажем, назначено три встречи с разными клиентами.

Все существующие программы-календари устроены так, что дедлайны и текущие дела в них не учитываются, а вместо этого все построено на встречах. Но встреча представляет собой однократное событие с определенным временем начала. Встреча – неотъемлемый компонент

тайм-менеджмента, но далеко не единственный. И дело не ограничивается отсутствием других типов записей для календаря, даже эти встречи имеют искаженное представление.

Обозначать время начала встречи куда как более важно, нежели ее окончание, но календари не делают разницы между этими двумя моментами. За последние тридцать лет я участвовал или назначал сам тысячи встреч. Невероятно важным является именно время начала встречи. А вот *время окончания* в большинстве своем такого значения не имеет, его не указывают и часто оно даже неизвестно. Но в каждой календарной программе, что мне попадалась, у всех встреч был атрибут времени окончания, который требовалось обозначить с той же тщательностью, точностью и усердием, что и время начала. Это значение времени окончания используется для вычисления точных часов и минут, в которые вы будете доступны для связи, что на самом деле не может быть обозначено достоверно и значительно искажает действительность. Представим, что вы отправите мне приглашение на встречу на три часа дня при помощи стандартной программы-календаря. Если у меня уже назначена другая встреча, к примеру, на 14:30, длительностью 35 минут, то программа отклонит ваше приглашение. На деле же я всегда могу улизнуть с предыдущей встречи на пять минут пораньше.

Помимо этого, ни в одной из подобных программ не учитывается время, за которое я должен добраться до назначенного места. Если у меня назначена встреча на другом конце города в 14:00, то мне нужно выехать на нее в 13:30. Какое время в этом случае мне внести в календарь – 14:00 или 13:30? Хорошо спроектированная программа должна учитывать этот момент и следить, чтобы я вовремя попадал куда нужно.

Помимо указанных, существует множество других распространенных типов записей, зависящих от времени, которые тем не менее не учитываются в календарях. Так, у меня может быть десяток или больше текущих дел в любой из дней, тогда как фактически в каждый период я работаю лишь над одним из них. Типичная программа-календарь не желает признавать мое вполне обычное поведение и не позволяет добавлять в расписание элементы, которые относятся к проекту. Я не могу разойтись с этим медведем-плясуном.

Массовая интернет-истерия

Всемирная паутина открыла впечатляющий мир интернета каждому обладателю компьютера и модема. Веб-пространство – это великолепный инструмент с фантастическими возможностями. Поражает то, что наиболее важной переменой, какую привнес интернет, стал способ показать, насколько простым может оказаться использование программного обеспечения. Значительная часть из первых апологетов Веба была так захвачена простотой его использования, что требовала той же простоты и от всех прочих программ. В особенности им импонирует то, что работа через браузер позволяет избежать нудного процесса инсталляции программ.

Топ-менеджеры в индустрии программного обеспечения, в особенности поставщики корпоративных ИТ-решений, настойчиво пытаются примкнуть к этому повальному увлечению. Они бросаются воспевать любовь к браузерным приложениям, потому что это позволяет им продвигать свои продукты и не мучить пользователей отвратительным процессом установки. До того как Всемирная паутина завладела миром, пользователям приходилось пройти через сложный процесс инсталляции программы, в то время как браузерным приложениям это не нужно. Подавляющее количество руководителей в сфере компьютерных программ считают это не меньшим технологическим прорывом, чем изобретение застежки-молнии.

Но ведь нас просто водят за нос! Нет никаких причин, почему бы любым небраузерным программам не обладать «бесшумным» процессом установки, безотносительно технологии их реализации. Если бы на ваш компьютер требовалось установить программное обеспечение, это произошло бы независимо от наличия на нем браузера. *Есть лишь одна причина, почему программы, предназначенные для работы вне браузера, требуют установки: их разработчики не представляют иного хода событий*. Отдавая часть вопросов на откуп программе установки, они упрощали себе работу. В ранних версиях браузеров задавать такие вопросы не было возможности, потому разработчики просто пожали плечами и исключили их. В дополнение стоит сказать, что разработчики едва ли обратили внимание на этот недостаток, но благодаря ему для многих обычных пользователей Веб стал самой легкой в использовании платформой из всех, что им когда-либо доводилось встречать.

Однако стоит нам отвлечься от вопросов установки, и мы увидим, что браузеры слабы, словно новорожденные котята. Их язык, на котором они говорят с пользователем, напоминает доисторический диалект. Их архитектура похожа на чью-то злую шутку. И они столь же гибкие, как сосульки. Любому приложению, заключенному в рамки браузера, непременно приходится жертвовать своими возможностями и невероятной мощью. Меня разрывает от ярости, когда я вижу, что творят руководители со своими программами: портируя приложение под Веб в стремлении освободиться от процесса установки, они тем самым просто вырывают ему сердце. Тогда как они могли бы достигнуть того же результата, если бы просто сказали своим разработчикам: «Будьте так добры, избавьте это приложение от процесса инсталляции!»

Пользователи так держатся за браузерные приложения, потому что более хороших альтернатив у них перед глазами нет. Чего не скажешь о разработчиках программ, которые следуют той же тенденции, руководствуясь неверными предпосылками. Организация веб-пространства похожа на Советский Союз: диктатура центральных компьютеров определяет действия беспомощных настольных устройств. Разработчики, преимущественно те, что работают в корпоративных ИТ-подразделениях, управляют центральными компьютерами и, в точности как советские комиссары, пытаются извлечь выгоду из своего положения. Вместо того чтобы безвозмездно предоставить пользователям программы-без-установки, программисты вынуждают их расплачиваться колоссальной потерей долговременного контроля над всей информационной инфраструктурой.

Что плохого в программном обеспечении?

В своей первой книге я по большей части описывал ответ на этот вопрос в деталях. Тем не менее мне хотелось бы посвятить еще несколько страниц этой теме и окинуть беглым взглядом некоторые принципы дизайна и проектирования, доказавшие свою эффективность при создании более качественных программ.

Программы имеют свойство забывать

Всякий раз при взаимодействии с программой вы запоминаете о ней чуть больше, но вот программа не помнит ничего. Трой Дэниелс, наш медийный продюсер, практически «живет» в Adobe Photoshop. Тем не менее каждый раз, когда он запускает программу, то видит, что она не запомнила ничего из всех его манипуляций с ней. Она не помнит ни места хранения изображений Троя, ни тех действий, которые он обычно с ними совершает. К тем элементам управления и командам, которыми он пользуется постоянно, отношение программы ровно такое же, как и ко всем прочим, которые он никогда не использовал и едва ли начнет.

Программы имеют свойство лениться

Большинство прикладных программ не направляют свои усилия на пользователей. Я не имею в виду, что они не работают, но они обычно так стараются угодить пользователю, будто бы перед ними программист. Это почти как подарить супруге на день рождения дрель. Тот факт, что вы сами в восторге от дрелей, совсем не значит, что она тоже придет в экстаз от такого подарка.

Если бы мы только могли побудить разработчиков создавать что-то действительно нужное пользователям, пользователи стали бы куда счастливее, а разработчики не выдумывали бы себе лишней работы.

Программы утаивают важные данные

Многие интерактивные продукты отличаются скупостью в предоставлении важных данных, как тот банкомат, что утаивал от меня количество денег на моем банковском счете. Они стараются замаскировать все, что на самом деле *происходит внутри них*, равно как и сокрыть нужную для понимания внутренних процессов информацию. Типичному пользователю интерактивной системы ни за что не догадаться, в каком состоянии она находится, до тех пор, пока она не огорошит пользователя сообщением о тотальном сбое. Яркий пример – мои новые радиочасы, которые я описывал в главе 1 «Загадки информационной эры». Они пытаются меня одурачить, ненароком скрывая свое состояние. Со стороны кажется, будто система функционирует нормально, хотя это не так, – только способа добраться до истины нет.

Если вдруг при работе за компьютером вы обнаружите себя черкающим заметки о функционировании программы на листе бумаги, знайте – вы стали жертвой программы, скрывающей данные. А ведь так просто добавить в любую программу отображение большего количества информации на экране, только мало кто из разработчиков задумывается об этом. Приведу пример. Когда мне приходит электронное сообщение через почтовый клиент, программа выводит крошечный значок конверта. Этот значок одинаков, вне зависимости от того, пришло ли в мой ящик одно сообщение или тысяча. Программа не дает мне ни малейшего намека на величину «стопки» входящих сообщений в моем цифровом ящике. Из-за такого сокрытия важной информации я не вижу картину целиком.

Программы лишены гибкости

Человек, получив шанс оценить ситуацию в целом, чаще всего имеет возможность подстроиться под происходящее, а вот программы редко обладают подобной гибкостью. Стоит офисному сотруднику обнаружить разросшуюся до 15 сантиметров в толщину кипу рабочих бумаг на его столе, как он тут же осознает, что пора принимать решительные меры, иначе завала не избежать. Структура программ в подавляющем большинстве случаев такова, что человеку виден лишь самый верх стопки, а все, что ниже, скрыто от его глаз. Продолжая аналогию с кипой бумаг – компьютер, вне зависимости от количества документов, в которых он «увяз», продолжит вести себя так, словно его решения ожидает только один верхний документ. Верна и обратная ситуация. Если в папке с входящими поручениями у сотрудника офиса только один документ, он может отложить его обработку и прийти на помощь коллеге, чья кипа бумаг гораздо больше. Компьютер никогда не сделает подобного.

Когда дело доходит до автоматизации ручного труда, разработчики (или аналитики) начинают исследовать присущее пользователям поведение при выполнении рутинных видов работ, а затем выделяют из этого функции и задачи. Затем они формализуют эти задачи в программном коде. Как правило, все, что не может быть формализовано, просто-напросто утрачивается.

В человеческой системе с ручной обработкой данных может статься так, что заявление близкого должностному лицу человека окажется на дне бумажной стопки, тогда он вытащит этот документ и ускорит процесс его обработки. А может быть и наоборот: заявление назойливого человека, грубо высказавшегося в отношении этого должностного лица в телефонном разговоре, может переместиться в самый низ стопки. Гибкость системы – залог поддержания порядка в обществе. Компьютерные системы обладают бесчувственной рациональностью, что отнюдь не благотворно влияет на прочность цивилизации.

Людям больше нравятся такие системы, в которых получается играть не совсем честно. Они хотят получить возможность несильно толкнуть автомат для игры в пинбол – совсем чуть-чуть, не для того чтобы нарушить весь ход игры, а чтобы слегка повлиять на ее результат в положительном ключе. Благодаря такой возможности «прогнуть» систему все то, где применим

ручной труд, работает более эффективно, чем в автоматизированной системе, хотя и более медленно.

Программы во всем обвиняют пользователей

Стоит программе обнаружить проблему, она неизменно свалит все на пользователя, да еще и обвинит его в возникновении этой проблемы. Если же подобное случается с человеком, он обычно пытается самостоятельно устранить последствия. Например, если я случайно задену чей-то бокал и разолью вино, будучи в гостях у знакомых, я быстро промокну пятно своей салфеткой, чтобы оно не растеклось, а потом наполню для гостя новый бокал. Ввиду той заботы и желания помочь, которые я проявлю в результате, инцидент удастся сгладить, и никто не будет держать на меня зла.

Как-то раз я пытался попасть на сайт техподдержки одной программы через ее собственный интерфейс. По какой-то неведомой причине программе не удалось установить соединение с сетью. Она выдала ошибку, в которой сообщалась ложная, оскорбительная и совершенно бесполезная информация о том, что у меня будто бы «отсутствует подключение к сети интернет». Это выглядело так, словно программа разлила мой бокал вина, отказалась принимать меры, да еще и обвинила в этом меня самого.

Если с проблемой, даже небольшой, сталкивается программный продукт, он часто просто впадает в ступор, не предпринимая никаких действий, становясь абсолютно бесполезным. Подобный сбой обычно тянет за собой ряд побочных проблем. К примеру, программа инсталляции перед тем, как начать копирование файлов на жесткий диск, задаст пользователю несколько вопросов. Раньше, если инсталлятору не хватало места на жестком диске, процесс установки просто прерывался. Однако современные программы не стали лучше в этом отношении. В тот момент, когда заканчивается свободное дисковое пространство, инсталлятор выводит сообщение об ошибке, но затем все равно прерывает процесс установки, не сохраняя все те параметры, которые вы так скрупулезно настраивали до начала процесса. Когда вы освободите место на жестком диске и вновь запустите процесс инсталляции, вам придется заново отвечать на те же вопросы программы, хотя она могла бы сохранить ваши предыдущие ответы.

Программы не желают нести ответственность

Диалоговые окна подтверждения представляют собой один из самых типичных примеров плохого проектирования – я говорю о тех, что спрашивают, *уверены* ли мы, что хотим совершить какое-либо действие. На заре становления персональных компьютеров действия пользователя были необратимы – программа немедленно выполняла введенную команду. Когда пользователь вводил команду erase all («стереть все»), программа моментально и безвозвратно удаляла данные. Не сомневаюсь, что после того, как кто-то из пользователей нечаянно удалил все данные с жесткого диска, он сразу же пожаловался на это программисту, и тот добавил в программу единственный и, как ему казалось, адекватный способ предупреждения таких инцидентов. Теперь, когда пользователь наберет команду erase all, компьютер, прежде чем выполнить ее, запросит подтверждение действию удаления.

И это столь же логично, сколь и некорректно.

Диалоговое окно подтверждения – весьма удобный способ для разработчика снять с себя ответственность за соучастие в случайном удалении. Однако мы неверно понимаем саму проблему. Ответственность за удаление данных полностью лежит на самом пользователе, а он уже ввел эту команду. Не время программе сомневаться в его действиях. Ей следует принять этот сигнал и выполнить ту задачу, которую ей поручили. Ответственность, от которой на самом деле уходит программа, – это готовность к *отмене* действий пользователя, даже если он отправил команду на исполнение.

Процесс принятия решения у человека, как правило, отличается от компьютерного, поэтому у людей считается нормальным и вполне типичным желание отменить ранее принятое решение или передумать. В мире реальных действий, не связанных с компьютерами, человек может отложить, изменить или обратить большую часть того, что сделано. Нет никаких причин, почему

так же не могут вести себя и программы, за исключением нежелания программистов поразмыслить над этим.

Банкомат, о котором я упоминал в главе 1, отказывается от ответственности точно так же, как и настольные приложения, – запрашивая подтверждения своим действиям. Стоит мне вставить карту, как он тут же требует подтвердить это. Стоит запросить наличные – он снова требует подтверждения. Стоит ввести сумму – просит подтвердить введенное число. Отчего бы ему просто не довериться мне? Что мешает ему безропотно провести транзакцию?

Он мог бы позволить мне прервать транзакцию в любое время более изящным способом. Для этого на нем просто должна быть большая красная кнопка с надписью «ОТМЕНА», тогда я мог бы воспользоваться ею в любой момент. Так банкомат дал бы понять, что считает меня умным, способным, осознающим свои желания и контролирующим свои действия, а не недалеким и некомпетентным человеком, не отдающим отчет в том, что ему нужно.

Не спорю, что некоторых пользователей такого банкомата *в самом деле* можно отнести к разряду недалеких и некомпетентных людей, но даже самому недалекому и некомпетентному человеку не понравится, если с ним будут обращаться как с таковым. Более того, если к клиенту относиться подобным образом, у него никогда не возникнет привязанности к продукту и позитивных эмоций.

Проблему исправить несложно. Программа должна вывести надпись «Снятие наличных» вверху экрана и *не убирать ее оттуда до завершения транзакции*. После этого на экране должен отобразиться процент за снятие наличных: «$1.50» и *тоже оставаться там* до следующей операции. Далее должна добавиться надпись «Проверка данных…», а ниже – номер моего банковского счета, текущий баланс и размер лимита на снятие наличных, – *информация должна остаться на виду*. Теперь, когда на экране появится запрос суммы, которую я хочу снять, я уже буду подготовленным информированным человеком, а вовсе не жертвой, подвергшейся допросу. Ответственное решение, которое я приму далее – о сумме денежных средств, будет основываться на понимании того, что доступно, возможно, законно и необходимо для меня.

Та система, что я описал выше, предоставляющая полезную информацию пользователю, являет собой типичный пример того, как функционируют человеческие системы, – людям просто необходимо видеть ситуацию целиком. Компьютерам же, напротив, нужны лишь небольшие порции информации для принятия решения о переходе к следующему шагу процесса. И именно на этом базируется проектирование взаимодействия: предполагается, что там, по другую сторону системы, находится еще один компьютер, а не живой, обладающий чувствами человек, давящий на кнопки на холодном ветру, пока его друзья нетерпеливо топчутся рядом.

*** * ***

Те, кто впервые сталкиваются с миром компьютеров, предполагают, что у программ есть уважительная причина для того поведения, каким они обладают. Однако все совсем наоборот – подобное поведение обычно появляется по чьей-то прихоти или в результате случайности, и такая беспечность продолжается из года в год. Но стоит вовремя добавить в процесс разработки толику дизайна и проектирования, как мы сможем изменить такое поведение на более эффективное и приятное для пользователя.

5. Нелояльность клиентов

Настоящая выгода, которую дает прекрасно спроектированное программное обеспечение, – это горячая приверженность вашему продукту, возникающая в клиентской душе. В данной главе вы увидите, как может подобная преданность помочь компании удержаться на плаву в самые нелегкие времена и успешно отражать натиск конкурентов. Также вы увидите, какой беспомощной окажется ваша компания без этой преданности.

Желанность

Ларри Кили из *Doblin Group* разработал занятную концепцию о трех важнейших качествах, необходимых для выживания в высокотехнологичном бизнесе. Первое качество Кили назвал потенциалом, в компании оно обеспечивается за счет инженеров. Они постоянно задаются вопросами: «На что мы способны? Каковы наши возможности?» Задача инженеров – точно

понимать, что возможно создать, а что нет. Продукт не станет успешным, если его никак не воплотить в жизнь и не обеспечить функционирование.

Название второго качества, по модели Кили, – жизнеспособность, которой наделяют проект предприниматели. Их вопросы звучат так: «Какой продукт будет жизнеспособен? Что у нас получится продать?» Предприниматели обязаны понимать, создание каких продуктов приведет к прибыли, а какие создать невозможно. Продукт не станет успешным, если его продажи не принесут достаточно средств для поддержания развивающейся компании.

А так как все успешные высокотехнологичные компании заинтересованы в поддержании баланса между этими двумя качествами, они очень крепко взаимосвязаны. Предпринимательская активность в значительной мере зависит от способности инженеров производить на свет продукты, которые работают. В свою очередь, инженеры целиком зависят от предпринимателей, поскольку те могут обеспечить их всем необходимым для работы. Такой симбиоз невероятно сложен.

Самая большая любовь программистов – добавлять в свои продукты всевозможные функции. Они с энтузиазмом примут творческий вызов и заставят ядро программы работать с максимальной отдачей. Так проявляется их потенциал, поэтому большинство технических специалистов останутся счастливыми, даже если им никогда не придется вывести на рынок жизнеспособный продукт. Если компания, в которой они работают, терпит крах, они легко находят новое место. Их личная успешность не зависима от успешности предприятия в целом.

Предприниматели, напротив, озабочены тем, чтобы завладеть долей рынка и совершить больше продаж. Их азарт заключается в том, чтобы замотивировать людей на покупку продукта. Так проявляется жизнеспособность, поэтому некоторые предприниматели останутся счастливы, даже если их продукт будет далек от технологического идеала. Большинство предпринимателей вполне устроит продажа камней-питомцев, до тех пор пока на них будет достаточно покупателей.

И хотя две эти стороны связаны друг с другом, разница в целях, к которым они стремятся, неизменно окажется слабым звеном их отношений. Их связь будет так же неустойчива, как двуногий табурет, и тогда в игру вступает еще одна сила – третье качество по концептуальной модели Кили, которое выполняет роль стабилизирующей третьей ножки табурета.

Третье качество, согласно Кили, – это желанность, за которую ответственны проектировщики. Вопросы, которыми они должны задаваться: «Что позволит привлечь людей? Чего они желают?» Задача проектировщиков – понять, как должен вести себя продукт, чтобы люди получали удовольствие от взаимодействия с ним и чувствовали себя счастливыми. Продукт не продержится на рынке долго, если будет не в состоянии дать ощущение силы и довольства своим потенциальным покупателям.

Проектировщики берут за основу продукт, который потенциально может быть произведен и способен функционировать, который можно реализовать и получить прибыль с продаж, и доводят его до полного успеха, превращая в то, что потребители *действительно захотят*. Третья ножка этого табурета помогает ему устоять, превращая просто достойные внимания технологические разработки в долгосрочный успех.

И хотя выявить нечто привлекательное возможно в уже существующем продукте, Кили утверждает (и в этом я его поддерживаю): более разумно *первым делом* решить, что покажется привлекательным покупателям, а уже потом организовать инженеров и предпринимателей на создание желаемого продукта и планирование его продаж. Разумный человек сразу сообразит, что в применении именно такого подхода есть огромные преимущества. Он позволит вам стать на голову выше конкурентов. Пока все они шпионят друг за другом, соображая, каким эффектным ходом лучше ответить на ход конкурента, бьются над вопросами «возможно ли?» и «жизнеспособно ли?», вы готовитесь для решительного марш-броска, фокусируясь на еще не удовлетворенных потребностях ваших потенциальных клиентов. Оставьте ваших конкурентов сражаться друг с другом, пока вы займетесь предоставлением вашим покупателям именно того, чего они желают больше всего.

Приведу пример: в начале 1990-х годов компания *Borland International* занимала одну из лидирующих позиций на рынке разработки программ для Windows, и я имел возможность изучить этот бизнес, пока был привлечен в компанию в качестве консультанта. Примечательно, что компании удалось объединить больших экспертов предпринимательского дела и невероятно

сильных профессионалов в разработке программного обеспечения. Буквально каждый день мне представляли очередной впечатляющий экспериментальный проект, разрабатываемый в строгой секретности. За каждый такой проект отвечали два специалиста: предприниматель высочайшего класса и столь же блестящий инженер-разработчик.

Проекты имели сходство по ряду моментов: прорывная технология, явное закрытие потребностей рынка, несомненный коммерческий потенциал, блестящая команда. Поначалу такое огромное количество талантливых людей, работающих над впечатляющими проектами, поражало воображение. Но спустя некоторое время обнажилась истинная природа этих проектов: лишь очень незначительная часть из них завершалась выпуском продукта на рынок. И ни один из них не имел представления о предполагаемых потребителях. Доходы с продаж этих продуктов были невелики, а расходы крайне расточительны; в итоге спустя пять лет перетягивания каната между потенциалом и жизнеспособностью компания *Borland* вполне предсказуемо вступила в тяжелую пору и была вынуждена сократить большинство персонала.

Компания *Borland*, как и многие другие современные высокотехнологичные предприятия, не имела в своем штате хоть сколько-нибудь примечательных специалистов по проектированию. В ее предпринимательской и технической культуре также отсутствовало глубокое понимание роли проектирования. Это и явилось причиной того, что для *Borland* стало тяжелым испытанием сделать их жизнеспособные продукты с широким потенциалом привлекательными для потребителя.

Желанность легко перепутать с потребностью (нуждой), однако они существенно различаются. Я *желаю* провести полтора месяца на Бермудских островах, но не испытываю *нужды* в этом. А вот если у меня обнаружили желчные камни, то я *нуждаюсь* в хирургическом вмешательстве, но это не является *желанным* для меня. Агенту по недвижимости Салли *нужно* продать четыре дома в этом году. При этом она *желает* обеспечить четыре семьи счастьем и уютом. Для продажи недвижимости ей *нужно* пользоваться MLS-программой, чтобы публиковать предложения в многочисленных каталогах, но она *желает*, чтобы эта программа не заставляла ее чувствовать себя глупой.

В краткосрочном периоде человек может попасть под сильное влияние собственных нужд, но в долгосрочной перспективе на первый план выходят его истинные желания, которые способны оказать более значимое воздействие. Человек всегда возвращается к своим желаниям, когда удовлетворены все нужды. Если Салли испытывает потребность в чем-то, она сделает необходимые действия для получения этого, но если она чего-то действительно *желает*, она будет этому безгранично преданна. Она отдает себе отчет в том, что тратит свои свободные средства, которые остались после уплаты обязательных расходов на базовые потребности, а

потому купит именно то, что делает ее счастливой, даже если эта трата не вполне рациональна. Если потребитель желает заполучить определенный продукт или бренд, его лояльность станет сильнейшим двигателем бизнеса.

Треугольник Кили показывает нам, как постоянно получать преданных клиентов. Компания – разработчик программного обеспечения вполне может быть *жизнеспособной*, удовлетворяя одни лишь потребности агента по недвижимости Салли. Но треугольник также показывает нам, что компания может стать сильнее, просуществовать дольше и занять лидирующие позиции на рынке, если будет *желанной* для Салли. В том случае, когда продукт просто закрывает базовые потребности Салли, она станет либо его приверженцем, либо противником, вынужденным молча терпеть неудобства. В обоих случаях, несмотря на необходимость обучаться использованию программы, она не будет ею довольна и не порекомендует ее коллегам. Однако если продукт удовлетворяет *желания* Салли, он станет ее другом и помощником в ежедневной работе. Салли станет фанатом, энтузиастом использования этого продукта. Она расскажет о нем коллегам и друзьям. Она будет счастлива при решении рабочих задач и испытает гордость от своей работы. Таким способом MLS-программа повысит эффективность труда и эмоциональный настрой Салли, пробудит в ней лояльность к продуктам компании.

Продукт, в который не заложена концепция желанности, сможет закрыть потребности рынка, однако каждый его успех будет успехом танцующего медведя. А самой большой слабостью таких продуктов является то, что чувства лояльности в потребителях они не вызывают. У компании без лояльных потребителей гораздо меньше шансов выдержать натиск конкурентов.

Небольшое сравнение

Динамику треугольника Кили можно явно продемонстрировать на примере трех известных высокотехнологичных компаний, обладающих своими достоинствами и недостатками: *Novell*, *Microsoft* и *Apple*.

Недостаток преданных клиентов в долгосрочной перспективе обычно приводит компанию в упадок, несмотря на сильные позиции в удовлетворении потребностей рынка. Компания *Novell*является ярким примером этого утверждения. В начале 1990-х годов создать офисную локальную сеть наиболее практичным способом можно было только с помощью программного обеспечения Novell NetWare. Программа NetWare – с точки зрения продукта – обладала несомненным потенциалом, а сама *Novell* – с точки зрения компании – несомненной жизнеспособностью. Локальные вычислительные сети (ЛВС) в то время нужны были всем, и ни один другой вендор не мог закрыть эту потребность. Было еще несколько компаний, например *Banyan* и *Corvus*, которым также удалось разработать достойные технические решения – таким образом, потенциал у них тоже был. Они уступали лишь в жизнеспособности, потому что их бизнес-модель не оправдала себя. Однако ни одна из этих компаний не была успешна в создании желанного для пользователя продукта, и, как следствие, невзирая на процветание компании *Novell*, клиенты пользовались ее продуктом NetWare для создания ЛВС только из необходимости быстро закрыть текущие потребности, так что эта программная система была просто нелюбимым пляшущим медведем.

Компания *Novell* росла вширь и лучилась счастьем, однако система NetWare была невероятно плохо спроектирована, отчего ее установка, модификация и сопровождение требовали участия опытного дорогостоящего специалиста. Более того, сети, построенные на основе NetWare, отличались грубым и неподобающим поведением, разочаровывая пользователей. *Novell* не смогла этого понять, возможно, потому, что программу NetWare покупали миллионы клиентов, только дело в том, что основной контингент заказчиков делал это из *нужды*, а не из *желания*.

В начале 1990-х годов такие компании, как *Microsoft, 3Com* и даже *Apple*, вывели на рынок собственные программные продукты, обладавшие сходным с NetWare потенциалом, но при этом над клиентами не тяготела зависимость от сторонней техподдержки и, что более важно, им не требовалось постоянное сопровождение. Компания *Novell*, словно в страшном сне, наблюдала, как стремительно падают ее позиции на рынке. С возникновением конкуренции недостаток преданных клиентов у *Novell* сразу дал свои «плоды». На текущий момент основное направление деятельности компании *Novell* заключается в сопровождении тех клиентов, которые по техническим причинам не смогли отказаться от ее услуг. Новые клиенты ушли к другим компаниям.

Компания *Novell* обладала жизнеспособностью и невероятным потенциалом. У нее была мощная технологическая разработка и серьезная ниша в бизнесе, но все ее беды случились из-за полного отсутствия проектирования взаимодействия.

* * *

История компании *Microsoft* проста и понятна. Продукты, которые она разрабатывает, технически конкурентоспособны, но инновационными их не назовешь. *Microsoft* предпочитает идти по уже проторенной дорожке, вслед за компаниями, проведшими собственные исследования и разработки[11]. Однако нельзя ставить под сомнение тот факт, что Билл Гейтс является одним из самых выдающихся предпринимателей своего поколения, а то и всего двадцатого века. Ему присущ удивительный талант добиваться успеха почти в каждом деле, какие бы препятствия ему ни попадались.

Компания *Microsoft* не занимается дизайном и проектированием вовсе или уделяет им недостаточно внимания, а ее продукты известны тем, что заставляют пользователей чувствовать себя глупыми. Также они знамениты тем, что большую пользу от них можно получить, только приложив усилия и разобравшись с огромным количеством функций.

Многие компании и профессионалы привержены программному обеспечению *Microsoft*, но большинство из них вынуждены идти на этот шаг из экономических соображений или потому, что рынок не может предложить им другие варианты. Мало какие компании способны предложить столь же функциональные решения, как *Microsoft*. Как бы то ни было, принимать экономическую необходимость за преданность клиентов было бы опрометчиво. У компании *Microsoft* не так много действительно лояльных к ней пользователей.

У *Microsoft* есть некоторый потенциал, но вместе с тем ей присуща поразительная жизнеспособность. *Microsoft* также обладает достаточной технологичностью и занимает превосходную рыночную нишу, что способно компенсировать отсутствие дизайна и проектирования в краткосрочной перспективе.

* * *

Проницательные руководители компаний понимают невероятную ценность лояльных клиентов. Корпорация *Apple* заслуженно известна своей одержимостью применять принципы дизайна и проектирования на всех уровнях. В каждом аспекте фирменного стиля *Apple*, маркетинга и выпускаемых ею продуктах ощущается исключительный дух дизайна. Невозможно пересчитать, скольких наград и почестей удостоилась *Apple*, но достаточно одного взгляда на ее программное обеспечение, устройства, упаковки, документацию, и даже на те вечеринки, которые компания организует на конференции MacWorld, чтобы заметить близость дизайна и проектирования корпоративному духу компании.

Благодаря такой приверженности дизайну и вниманию к нюансам взаимодействия *Apple*обзавелась лояльностью клиентов, которая граничит с фанатизмом и часто им и является. Владельцы устройств Macintosh – самые преданные из всех пользователей каких-либо программных продуктов в мире. Ни один другой продукт или производитель не выводит личную преданность на такой высокий уровень, как это делает *Apple*. Клиенты компании наклеивают логотипы *Apple* на автомобили, заказывают специальные рамки для номерных знаков, носят футболки *Apple* и всяческими другими способами показывают свою принадлежность духу *Apple*. Они готовы превозносить достоинства «маков» любому, кто окажется рядом. И хотя в большинстве случаев компьютер Wintel способен удовлетворить все потребности пользователя лучше, быстрее и за меньшие деньги, чем Мак, для фанатов Macintosh всегда остается единственным возможным вариантом. На недавней конференции по дизайну компьютер Wintel вместо Macintosh был только у одной докладчицы, и при этом она приносила ярые извинения аудитории в связи с «предательством» единственного компьютера, которым должен обладать всякий, у кого есть хотя бы минимальное чувство стиля.

Технологическое мастерство *Apple* обладает величием, но гениальности в нем нет. С точки зрения инновационного потенциала *Apple* не превосходит *Microsoft*.

Те лидерские позиции, которые *Novell* потеряла за год, компания *Apple* теряла 12 лет. И лишь немногие проблемы *Apple* были вызваны внешними факторами. Больше всего компания пострадала от великого разнообразия проблем, которые сама же себе и создала. К примеру, в середине 1980-х Стив Джобс – главный основатель и идеолог компании – был изгнан и впоследствии заменен исполнительным директором, не имевшим никакого отношения к компьютерной сфере, употреблявшим только безалкогольные напитки и раз за разом принимавшим некомпетентные управленческие решения. Цены на продукцию были завышены, а маркетинг оказался очень плох. К сторонним разработчикам программного обеспечения выказывалось снобистское, пренебрежительное отношение. И, помимо прочего, «мак» продолжал оставаться закрытой системой – именно эту стратегию массово считают причиной свержения других платформ, лидировавших на рынке (таких, как VMS, MVS и OS/2).

Указанные ошибки и промахи с легкостью привели бы к краху обычную компанию, но умелое использование корпорацией *Apple* дизайна и проектирования, благодаря чему Макинтош стал желанным приобретением, помогли *Apple* обрести неслыханную преданность клиентов. Компьютеры Mac закрывали потребности пользователей практически так же, а в некоторых моментах даже хуже, чем это делали устройства под управлением Windows, но удовлетворение базовых нужд потребителей не является тем, что приведет продукт к успеху на рынке.

Продолжительная критика в адрес руководства, ощутимые финансовые потери, создание продуктов среднего качества, миллиарды, впустую потраченные на исследования, потеря двух третей рынка – ни один из этих факторов не мешает *Apple* иметь преданную армию клиентов, чем не может похвастаться ни одна другая компания. Благодаря этому компания получила огромные преимущества в ведении бизнеса. Многие из них сложно измерить, и ни одно из них не отражается в финансовых показателях компании, однако они столь же реальны и ценны для держателей акций, как чек на дивиденды.

Благодаря потребительской лояльности, вызванной повышенным вниманием *Apple* к дизайну и проектированию, истинные поклонники компьютеров Mac готовы игнорировать преимущества, во множестве предоставляемые продуктами других компаний. Тот факт, что приверженцы *Apple*отказываются от использования устройств другой марки, позволяет корпорации выиграть время в конкурентной борьбе и подготовить свой ответ на чужие инновации. За счет лояльности потребителей *Apple* способна пережить те внезапные изменения, которые преподносит очередной технологический прорыв. Компания *Novell* начала терять свои позиции в тот самый момент, когда ее конкурент – компания *Microsoft* – вывела на рынок собственный жизнеспособный продукт по управлению сетями. Обладание значительной долей рынка *Novell* не смогло уберечь ее от натиска рыночных сил. Обратная ситуация у компании *Apple*, доля компьютерного рынка которой никогда не превышала 15 %, однако позволяет ей стойко сопротивляться атаке более мощных и дешевых компьютеров конкурентов.

Apple относится к той категории компаний, продукты которой являются желанными для пользователей. Благодаря приверженности компании дизайну, *Apple* смогла преодолеть невзгоды, связанные со средним качеством технологии и губительной политикой ведения бизнеса.

Если бы *Novell* добавила проектирование к иным своим сильным сторонам, у компании был бы шанс преодолеть последствия слабых управленческих решений. Стоит *Microsoft* опомниться и обратить внимание на важность проектирования взаимодействия, как все, что останется конкурентам, – это признать поражение и разойтись по домам. *Apple* сама разрушала себя изнутри, подобно звезде гранж-рока, но, появись у нее возможность преодолеть последствия таких деяний, ее жизнеспособность снова придет в норму.

Когда студенты бизнес-школ Гарварда и Стэнфорда изучают кейсы компаний, редко в каких случаях им говорят о значимости проектирования взаимодействия. Такое проектирование являлось обязательной составляющей успеха продуктов еще в прежней, индустриальной эре, несмотря на более простой подход к нему в те времена. Помимо этого, продукты индустриальной эры старше, их проблемы и пути решения этих проблем давно исследованы. В информационную эпоху – эпоху быстрых инноваций и крайне интенсивного когнитивного сопротивления – проектирование взаимодействия становится первоочередной необходимостью.

Пора выходить на рынок

В тот момент, когда какая-либо компания первой объявляет, что вывела на рынок продукт с востребованным функционалом, не следует сломя голову выводить эквивалентный продукт. Вы уже проиграли ей во времени, и одна лишь спешка не приведет вас к победе. Но у вас все еще есть отличная возможность выбить себе лидерские позиции, сделав более качественно спроектированный продукт, чем у конкурентов. Дизайн и проектирование сделают ваш продукт желанным, так что ваши потенциальные покупатели захотят владеть именно таким приобретением, игнорируя продукцию других компаний, пусть даже те и были первооткрывателями на рынке.

Той компании, которая первой выходит на рынок, обычно приходится чем-то жертвовать. И шанс, что этим окажется дизайн и проектирование взаимодействия, очень велик[12]. Поэтому, несмотря на быструю экспансию рынка, такая компания очень уязвима в отношении дизайна своего продукта.

Однако попытка снабдить продукт дополнительным новым функционалом – это отнюдь не то же самое. Новые опции не так важны потребителям, как адекватное взаимодействие с продуктом и те конкретные проблемы, которые можно решить с помощью базовых функциональных возможностей, так что добавление новых опций не приведет к столь же эффективному результату. В ситуации с рынком, который заполонили плохо спроектированные продукты, опции не дадут вам захватить большой рыночный сегмент[13].

На многих рынках присутствует множество продуктов, схожих по своей сути, при этом никто из создателей этих продуктов не задумывается о проектировании взаимодействия. Эти продукты конкурируют, лишь непрерывно меряясь дополнительным функционалом. Стоит одному разработчику объявить о новой опции, как все прочие тут же добавляют такую же опцию в последующие версии своего продукта. Такие рынки отличаются тем, что для них характерна раздробленность на множество мелких сегментов. Нет какого-либо доминирующего продукта или производителя. К примеру, на рынке программного обеспечения, выполняющего роль персонального информационного менеджера (Personal information manager – PIM), присутствует более десяти производителей. Аналогичная ситуация на рынке мобильных телефонов.

Битва между потенциалом и жизнеспособностью может длиться еще много лет, удерживая потребителей в напряжении. Лишь одна сила способна превратить раздробленный рынок, на котором безраздельно властвуют опции, в рынок более стабильный, ориентированный на дизайн и проектирование, – и эта сила должна прийти извне. И такой внешней силой в равной степени может быть как деловая хватка Билла Гейтса, так и продуманное применение дизайна и проектирования.

Невзирая на все усилия Билла Гейтса, его программные продукты по-прежнему не становятся желанными для потребителей. Более того, практически все высокотехнологичные продукты других разработчиков в среднем остаются на том же уровне желанности, что и продукты *Microsoft*, несмотря на ум, искренность и весь тот тяжелый труд, который вложили в них создатели. В следующем разделе я покажу, что простые повсеместно распространенные изъяны в процессе проектирования продуктов приводят к еще большему распространению неприятных и нежеланных программ, похожих на медведей-плясунов.

Часть III. Вилка для супа

6. Психбольница в руках пациентов

Несмотря на различную природу устройств, описанных в главе 1 «Загадки информационной эры», всех их объединяет одно: они вызывают у пользователя раздражение и неприязнь. В этой главе я продемонстрирую, что причиной такой распространенной ситуации является невольный перехват власти в индустрии техническими специалистами. Невзирая на возможные дискуссии в отношении маркетинга, стоит признать, что вид потребительских продуктов фактически определяется теми, кто наименее в этом сведущ.

Серый кардинал

Статья Мишель Куин[14], посвященная эффектному провалу высокотехнологичной компании *General Magic*, весьма наглядно описывает это явление. Упоминая, как президент компании Марк Порат «направил все усилия своей команды инженеров на проектирование устройства мечты», автор нечаянно затрагивает самую суть проблемы провала этого продукта. В

словах Мишель Куин нет ни капли иронии. Кажется вполне естественным для инженеров заниматься дизайном и проектированием продуктов, но в этом и есть корень проблемы. Далее в статье она приводит высказывание одного из инженеров: «Мы так до конца и не поняли, что это был за продукт. Мы составили спецификацию, только когда до завершения проекта уже оставалось около 8–12 недель». И снова ирония ситуации неочевидна ни для самого инженера, ни для автора статьи. В статье словно бы говорится, что все могло закончиться для *General Magic* гораздо лучше, если бы только инженеры составили спецификацию на месяц раньше.

Нет особой разницы, в какой момент процесса разработки возникают спецификации, – они в любом случае не могут заменить проектирование взаимодействия. Не важно и то, как сильно стараются разработчики, – они все равно не смогут спроектировать взаимодействие должным образом. Они не подходят для этой работы не только из-за качественно других методов выполнения задач, опыта и направления подготовки, но и из-за конфликта интересов между потребностями пользователей и попытками сделать свою программистскую жизнь проще. И все равно компании раз за разом позволяют программистам контролировать процесс разработки, при этом обычно от начала и до конца проекта. В некоторых случаях степень этого контроля можно четко отследить, но в большинстве своем это влияние осуществляется косвенно.

Мне как-то довелось своими глазами наблюдать такой ненавязчивый контроль в одной очень успешной, среднего размера компании в Кремниевой долине. На встрече присутствовал президент компании – очень здравомыслящий, маститый бизнесмен-основатель – вместе со старшим программистом, который отвечал за создание продукта. Президент представил нашему взору продукт и продемонстрировал его внушительные возможности. Назначение продукта было нам понятно, как понятно было и то, что все его возможности будет весьма затруднительно использовать ввиду слишком сложного интерфейса продукта. Мы с моей командой проектировщиков сразу же поняли, что программисты «проектировали» его по ходу дела, приблизительно в той же манере, как бобры «проектируют» свои плотины.

Президент выразил недовольство по поводу того, что его рынок отвоевывает конкурент, чей продукт куда как слабее. Однако он не смог пояснить, почему так происходит, – знал лишь, что продукт его компании превосходит конкурентов по уровню возможностей. И хотя президент специально прибегнул к нашим услугам для помощи в конкурентной борьбе, он наделил старшего программиста полномочиями поступать так, как тот сочтет нужным. Мы отчетливо понимали, что продукт крайне нуждается во внесении поведенческих изменений, и представили свое видение ситуации им обоим. С нашей точки зрения, это была типичная несложная задача перепроектирования продукта, которая обычно занимает несколько месяцев и приводит к тому, что продукт становится более полезным, практичным, мощным и приятным – то есть более конкурентоспособным. Тем не менее от последовавшей просьбы старшего программиста *не вносить какие-либо значимые изменения в процесс взаимодействия* мы просто впали в ступор. Он считал, что с этим и так все в порядке. Он думал, что проблемы с рыночной реализацией вызваны тем, что менеджеры по продажам недостаточно хорошо разобрались в возможностях продукта. Он хотел, чтобы мы разработали ряд маркетинговых материалов для изучения внутренним персоналом компании с целью увеличения эффективности продаж. Он упрямо отказывался признавать наличие недостатков в самом продукте, невзирая на то, что это неопровержимо подтверждалось наступлением «отсталого» конкурента.

Разработчики обычно так долго и усиленно изучают программное обеспечение, что тому старшему программисту казалось непостижимым, почему пользователи не желают уделить время попыткам разобраться в его творении. Он был готов согласиться, что в компании есть проблема, но отрицал свою причастность к ней. Вся вина, по его мнению, лежала на продавцах, которые не помогали пользователям разобраться в продукте. Он хотел, чтобы работа по решению проблемы заключалась, например, в том, чтобы разработать новый комплект обучающих материалов, но при этом он абсолютно не хотел замечать ни малейшего намека на его вину в провале продукта.

Самонадеянность этого программиста просто поражала. Он не только был ослеплен собственной гордостью от создания такого мощного продукта, но и вводил этим в заблуждение своего руководителя – президента компании, скрывая свою неспособность спроектировать продукт так, чтобы пользователи были счастливы.

Продукт компании был первым в своем роде, пионером нового метода сопровождения производственных систем. Компания стремительно развивалась, ей благоволили на Уолл-стрит, а два года назад она невероятно успешно вышла на IPO. Деловая пресса писала о ней только хвалебные отзывы, а общественные и бизнес-организации щедро осыпали ее наградами. Создавалось впечатление, что компания находится на верном пути, а величина ее рыночной капитализации только подтверждала это предположение.

Однако подобный успех является объектом пристального внимания не только конкурентов, но и инвесторов, партнеров и других заинтересованных лиц. Конкурентам компании был ясно виден потенциал рынка, но не менее ясно они видели и слабые места в продукте этой компании. Они видели мощь и большой набор опций продукта, но еще они понимали, что это просто пляшущий медведь. Продукт предоставлял новейшие возможности, но не сделал пользователей более счастливыми. Медведь плясал, но его пляски не радовали взор. Не требуется большого ума, чтобы в такой ситуации не заметить уязвимое место в продукте компании, потому конкуренты просто-напросто скопировали большую часть опций, но при этом добавили ему легкости в обращении. Теперь управленческие отчеты, которые генерировал продукт, были динамическими и понятными для руководителей, в то время как отчеты в исходном продукте формировались невразумительно и были статичны. Возникший из ниоткуда конкурент разом отобрал у компании 60 % рынка и сделал это при помощи продукта, в разы уступающего исходному по мощности!

Президенту компании помешал его прошлый опыт разработки. Изначально этот опыт помог ему создать столь мощный продукт, но теперь оказался помехой на его пути, заслоняющей еще более серьезное препятствие в виде старшего программиста. Президент так крепко сросся с нравственными устоями высокоинтеллектуальных разработчиков, что подобное поведение старшего программиста в его глазах не выглядело чем-то неподобающим, в то время как наша команда крайне недоумевала. Президент на самом деле не имел должного влияния. Всеми делами компании, словно серый кардинал, заправлял его старший программист.

Замышляя недоброе

Как-то я спросил своего коллегу Скотта Макгрегора, известны ли ему примеры компаний, в которых проекты по разработке вышли из-под контроля ввиду недостаточного внимания к проектированию взаимодействия, и он прислал мне одну историю, изложенную ниже. История эта весьма трагична, а еще более печальной ее делает то, что она так типична для нашей индустрии.

Как видно из этой прекрасно описанной истории, Скотт невероятно талантлив. Еще он является опытным проектировщиком с великолепной подготовкой – как теоретической, так и практической – в сфере разработки программного обеспечения, дизайна и проектирования. Его история начинается с того момента, как он присоединился к небольшому стартапу в Кремниевой долине, существующему на средства венчурных инвесторов. Основатели стартапа также обладали приличным опытом, в частности они несколько лет успешно работали в *Apple*. В один прекрасный вечер Скотт пригласил меня познакомиться с основателями и представить мою компанию. Президент и вице-президент компании по техническому направлению продемонстрировали нашей команде проекты, над которыми работала компания, – мы были крайне впечатлены. Идея продукта была отличной. В его основе лежала разумная доля хороших технологий, которые служили для удовлетворения весьма насущных потребностей рынка. У них было все для достижения успеха, за исключением верного подхода к проектированию взаимодействия. Вот история, которую поведал мне Скотт (рассказанная его собственными, великолепно написанными словами):

Президент компании заявил нам, что мы опередим конкурентов благодаря нашей скорости и сообразительности, и дальше с гордостью представил свою стратегию «Готовься, огонь, цельсь!», намекая на то, что, пока другие только-только наметят цель, мы уже давно выстрелим и тем самым добьемся успеха раньше всех. Ну мы и выстрелили, хотя сразу было понятно, что попадем себе же по ногам.

Конечно, мы успели сдать релиз 1.2 к 31 декабря, только под этим словом мы договорились понимать то, что будет готово 31 декабря к пяти вечера, чем бы оно ни оказалось. Программисты

работали безо всяких спецификаций. Внезапно мы поняли, что в программу нужно внести существенные изменения, хотя ничто до этого не предвещало, и случилось это 29 декабря.

Незадолго до этого я предлагал воспользоваться принципами проектирования. Я говорил, что нам нужно выявить всех ключевых пользователей и других заинтересованных лиц, описать их профили, а потом разработать тезисы их целей и тех задач, которые им понадобится выполнить для достижения этих целей. Затем, на основании указанных задач, мы бы могли создать визуальные представления ключевых объектов и описаний поведения взаимодействия. И после этого сразу же приступили бы к разработке продукта.

Но увы, руководство пришло к выводу, что мы не можем себе этого позволить. У нас не было на это времени. Вместо этого мы объездили множество клиентов, и везде наш президент рассказывал про нашу грандиозную затею. Люди были в восторге от идеи, но хотели больше конкретики. Говоря о конкретике, каждый из этих потенциальных пользователей преследовал свои интересы, и все они были различны. Один хотел получить продукт, отвечающий целям отдела продаж, второму требовался инструмент для работы с независимыми посредниками, третьему – для работы с клиентами. Кому-то нужно было управлять многочисленными документами, другому – веб-страницами и т. д., и т. п. С каждым следующим потенциальным пользователем описание релиза 1.2 все расширялось, превращаясь в набор опций, какие только можно вообразить.

Еще более печально то, что пользователи рассказывали только о тех опциях, которые хотели бы получить, но совсем не упоминали те функциональные возможности, которые уже были в их программах или браузерах и которые они принимали как должное. Ввиду того что об этих уже существующих возможностях речи не шло, они не были добавлены в спецификацию продукта, а потому не были реализованы.

У наших свеженанятых вице-президентов по продажам и маркетингу неделями не получалось заставить продукт работать на их компьютерах. Когда же наконец им это удалось, то во время работы программы с их компьютеров по несколько раз на дню исчезали данные. Производительность продукта падала все больше и больше. В демонстрационной версии, где обращение к данным выполнялось не более 100 раз, производительность была низкой, но приемлемой, а при других параметрах программисты систему не тестировали. В реальных условиях вызов данных осуществлялся более 1000 раз, отчего система функционировала откровенно медленно, со скоростью улитки.

Программа содержала три основных экрана, однако чтобы выполнить простое действие редактирования документа, приходилось переключаться между ними по несколько раз. Многие отдельные задачи требовали от пользователя не менее десятка кликов мышью, открытия и закрытия множества окон и постоянных переключений между мышью и клавиатурой.

В итоге все вылилось в то, что продукт был сложен для изучения, непригоден с точки зрения понимания и производительности, ненадежен и регулярно удалял данные. Продукт был напичкан всевозможными «уникальными» опциями, но не обладал базовой функциональностью, присущей всем типовым решениям в этой области.

Как и следовало ожидать, к концу февраля правление предприняло соответствующие меры, после чего президент компании и вице-президент по техническому направлению покинули свои посты.

Разумеется, это лишь краткий эпизод из всего произошедшего. Могло бы показаться, что это частный случай, не доведись мне многократно сталкиваться с подобным в компаниях, где я успел поработать за последние двадцать с лишним лет.

Я понял такую вещь: ваш результат всегда определяется только теми критериями, которые вы установите, и теми рисками, которые вы за это понесете. Вплоть до января все, чем руководствовалось наше правление, были только сроки и обещанные опции. Они никогда не устанавливали хотя бы минимальных критериев качества (вроде средней наработки на отказ, процента повреждения данных и т. д.), так что качество принесли в жертву. Не было критериев и для производительности (вроде среднего времени отклика программы после нажатия клавиши), потому скорость реагирования системы была непредсказуемой. Не измерялось и то, сколько времени нужно на изучение опции или насколько часто пользователь будет пользоваться функциональными возможностями, не допуская ошибок; таким образом, изучаемость и юзабилити также были принесены в

жертву. Но те критерии, которые были установлены – график сдачи и список опций, – были достигнуты, а ввиду того что этот список не содержал полноценного описания указанных опций, многие из них были реализованы лишь номинально.

Рассказ Скотта подтверждает общеизвестную истину: «Что посеешь, то и пожнешь». Если постоянно напоминать разработчикам про сроки, проект уложится в график, а если игнорировать потребности пользователя, то люди будут чувствовать себя униженными. Скотт продолжает рассказывать свою историю:

Инвесторы неустанно твердят, что у них не такая огромная куча денег, чтобы попусту растрачивать их на продукт, который не удастся продать, потому обязательно нужно изучить потребителей, прежде чем приступить к разработке. Руководители разработки, в свою очередь, свято верят в то, что времени и денег недостаточно на проектирование взаимодействия – проектировать можно до бесконечности и из-за этого остаться совсем без денег еще до начала этапа программирования. И потому они предпочитают создавать все новые и новые продукты вместо того, чтобы улучшать проектирование взаимодействия в имеющихся, и так продолжается, пока не закончатся деньги.

Со стороны последние несколько месяцев все казалось похожим на старый сумасбродный комедийный фильм или мыльную оперу, но без капли романтики. И я это по-своему ценю. Конечно, если бы я относился к ситуации только как к безумию, я бы так подробно все не расписывал. Но я душой горю за это дело. Я считаю, что нашим моральным долгом является прекратить впустую растрачивать время человеческой жизни на бесполезные занятия.

В своей книге «Об интерфейсе» ты упоминал, насколько важным является перестать тратить время пользователя. Я обеими руками «за». Но это только верхушка айсберга. Это происходит лишь после того, как продукт уже вышел на рынок и предлагается к продаже. Но ведь еще великое множество продуктов даже не доходит до рынка или проваливается на этапе продаж. Каждый инженер, которого я когда-либо встречал, глубоко переживал из-за опасения, что продукт, над которым он работает, может никогда не попасть в руки пользователя. И когда в результате разработку продукта закрывают или он терпит крах из-за недостаточно качественного проектирования взаимодействия, получается, что усилия инженеров пропадают впустую. А в мире не так много действительно хороших разработчиков, чтобы так бездарно растрачивать их время. В этом я и вижу наш моральный долг – не просто «прекратить растрачивать время пользователя», а не тратить впустую время и жизнь любого человека, включая разработчиков.

Как же больно было видеть ситуацию, словно Кассандра, – предугадывать неминуемую гибель и оставаться неуслышанным, лишь безмолвно наблюдая за возможностями этой гибели избежать. Я сделал вывод, что обучение методом проб и ошибок является настолько эффективным, что делает человека, прошедшего через него, абсолютно невосприимчивым к аргументам, базирующимся на фактах и цифрах.

Хотелось бы подчеркнуть, что опыт Скотта не так уж необычен. Другому моему коллеге из этой индустрии – Джону Ривлину – тоже есть что рассказать. У Джона своя небольшая, но достаточно успешная компания по дизайну и проектированию в Пало-Альто. Вот его история:

Перед стартом каждого проекта по разработке программного обеспечения мы всегда создавали детализированные планы проектирования. Этот случай был не исключением. Проект начался с того, что мы разработали спецификацию объемом в пятнадцать страниц, в которой описывалось пользовательское взаимодействие нашей будущей программы. В этой спецификации были и общие положения по проекту, так что мы могли отойти от изначального описания «одним предложением». Я придаю этому особую важность, поскольку мы работаем по фиксированной ставке, закладывая в нее возможные, по нашей оценке, риски.

Мы предварительно согласовали идею создания спецификации с руководителем проекта, ответственного за разработку в компании нашего клиента, после чего достигли договоренности об определенной цене. Спецификация была разработана и направлена начальнику этого руководителя проекта – техническому директору. На что мы получили такой ответ: «Почему на подготовку спецификации ушло так много времени? Вы потратили значительную часть заложенного на проект бюджета. Мы здесь спецификациями не занимаемся. Мы просто идем и работаем». В дальнейшем мы выяснили, что технический директор представлял себе функциональность существенно иначе, чем руководитель разработки. И понять это нам удалось лишь благодаря той самой «бессмысленной спецификации», но даже это не было достаточно

убедительным фактом в пользу проектирования программного обеспечения. И это был технический директор публичной компании из сферы высоких технологий, с ежегодным оборотом более ста миллионов долларов. Сумасшедшим домом и в самом деле заправляют безумцы.

И хотя все опасения руководителей разработки, связанные с проектированием, иррациональны, они все же обычно исходят из личного, абсолютно реального опыта. Когда-то руководители предпринимали попытки создать более совершенные продукты, для чего обращались к своим программистам за помощью в проектировании взаимодействий, что в итоге приводило к весьма плачевным результатам. Люди, не знакомые с методами проектирования, склонны соотносить потребности пользователей со своими, ставя себя на их место, так что программисты тоже попались на этот крючок. Любая команда разработки, проектирующая продукт на основании собственных предпочтений, потратит неимоверно много времени в попытках прийти к единому мнению относительно пользовательского взаимодействия, потому что ни у кого из них нет твердого понимания пользователей, так что процесс может тянуться бесконечно.

Компьютеры и люди

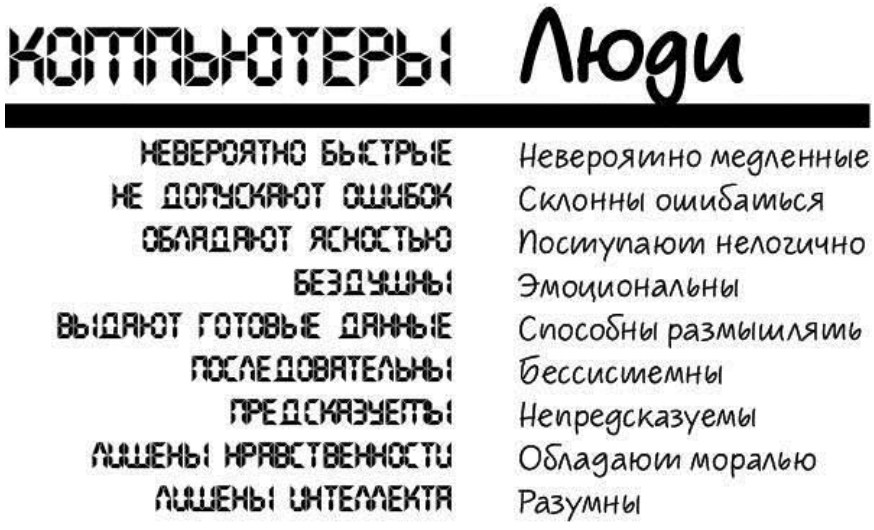
Программы больше напоминают мосты, нежели здания. Несмотря на то что они работают на основе высокотехнологичных микропроцессоров, ими должны суметь воспользоваться простые смертные. Среди всего этого внимания к новым технологиям и восхищения ими от нашего взора ускользает огромная разница между компьютерами и людьми, которые будут их использовать.

Так, ввиду того что у компьютеров есть память, мы полагаем, что она функционирует аналогично человеческой, хотя это далеко не так. Память компьютера работает так, как если бы он был пришельцем в мире людей. Благодаря моей памяти я могу с легкостью узнавать лица друзей, а вот мой компьютер никогда не узнал бы даже меня самого. Память моего компьютера с предельной точностью хранит миллион телефонных номеров, а я не всегда могу вспомнить даже собственный номер.

Чтобы программы могли обладать мощностью и надежностью, они должны быть созданы с учетом требований кремниевых микросхем. Чтобы программисты могли называть себя профессионалами, они также должны подчиняться этим требованиям.

Чтобы пользователи могли быть довольны программами и эффективно выполняли с их помощью свои задачи, программы должны быть созданы с учетом требований человеческой природы. Очевидно, что проблема кроется в кардинальном различии между этой человеческой природой и кремниевой микросхемой.

Не менее очевидно также то, что одна часть программы – внутренняя – должна быть написана на основе экспертных знаний технических нюансов и с учетом требований компьютеров. И так же понятно, что другая часть программы – внешняя – должна быть написана с вниманием к потребностям людей. Согласно моему твердому внутреннему убеждению, с первой задачей хорошо справятся программисты, а вот вторая – удел проектировщиков взаимодействия.



Джерри Вайнберг, гуру в компьютерной сфере, как-то сказал: «Устраняя вашу главную проблему под номером один, вы выпускаете на свет проблему под номером два[15]». В течение многих десятилетий главной проблемой под номером один в индустрии компьютеров оставалась проблема производительности. Компьютеры были, условно говоря, небольшими, дорогими, медленными и слабыми. Мы превозносили хакеров, словно божественных созданий, потому что они умели создавать программы с предельной эффективностью и выжимать максимум производительности из дорогостоящих мейнфреймов. По существу, было куда как менее затратно научить людей справляться с непонятным, но зато производительным программным обеспечением, чем тратить дополнительные средства на большее количество компьютеров. Однако тенденция к неизбежному снижению стоимости компьютеров фактически свела эту проблему на нет. Сегодня приспособление пользователей к такому «эффективному» программному обеспечению обходится в разы дороже, чем разработка программ, действительно отвечающих потребностям людей.

Выход достаточно очевиден: поставить программы на службу людям. Но тут наперекор нам встает культура, которую мы сами так заботливо пестовали в течение пятидесяти лет, превознося хакеров и вручая им бразды правления. При этом сообщество разработчиков программного обеспечения в большинстве своем согласно принять проектирование взаимодействия как один из этапов процесса разработки. Они говорят примерно так: «Конечно, проектируйте сколько душе угодно, но только дождитесь, пока мы свою работу закончим». Увы, проектирование взаимодействия должно выполняться перед этапом разработки, так что подобная «готовность» программистов к внедрению проектирования оказывается крайне безрезультатной. Это все равно что оператор бетономешалки вдруг заявил бы, что плотники могут строить каркас фундамента, после того как он закончит заливать бетон.

Обучаем собак кошачьим повадкам

В запасе у программистов всегда есть желание самостоятельно освоить проектирование. Ко мне беспрестанно обращаются с просьбами «научить проектировать». Меня восхищает такая широта взглядов, однако в эффективность такой затеи я верю мало. Каждый разработчик программного обеспечения, который считает себя достаточно хорошим, чтобы носить звание профессионала в этой области, слишком сильно погружен в ту символьно-детерминированно-последовательную логику поведения, столь присущую кремниевым микросхемам. И погружение это настолько глубокое, что становится невозможным одновременно достичь такого же значительного эффекта в иррационально-непредсказуемо-эмоциональном мире людей. Я вовсе не имею в виду, что программист не сможет стать проектировщиком; я лишь хочу сказать, что фактически невозможно выполнить оба задания с равной долей успеха, если делать их одновременно.

Каждому разработчику программного обеспечения свойственно считать себя особенным, единственным, кто способен выполнить обе задачи сразу. На самом деле это далеко не так, и печальная судьба *General Magic* ясно иллюстрирует это. Отдел разработки в *General Magic* возглавляли Билл Аткинсон и Энди Херцфельд. Оба они когда-то были ведущими разработчиками программного обеспечения для Apple Macintosh и, бесспорно, значительно превосходили многих программистов по уровню своего таланта, креативности и изобретательности. Их совместная работа над проектированием и программированием Macintosh привела к успеху в 1984 году (хотя Джеф Раскин, не участвовавший в программировании, также внес значительный вклад в проектирование). Тем не менее спустя 14 лет после их успеха мир уже не был прежним, и их методы работы уже не подходили. В начале 1993 года я взял интервью у Энди Херцфельда в головном офисе разработки *General Magic*, который располагался в гостиной дома Энди в Пало-Альто, – тогда он и поведал мне философию его взгляда на проектирование и программирование. Я слушал с удивлением, осознавая, как мала вероятность успеха этой теории. Однако история дала второй шанс проявиться выдающемуся таланту Энди.

Нет никаких сомнений в том, что продукт, который задумали в *General Magic*, был и до сих пор остается чрезвычайно желанным. Нет никаких сомнений в превосходстве применяемой технологии. Никто не может усомниться в превосходной способности Марка Пората создавать

стратегические партнерства и заключать выгодные сделки. И также несомненно то, что компания была основана серьезными людьми с серьезным финансированием. Что же в таком случае привело ее к краху? На мой взгляд, все указывает на проектирование взаимодействия, вернее, на его отсутствие. Невзирая на звездный состав и невероятную одаренность специалистов компании, продукт *General Magic* появился в результате процесса *конструирования*, а не *проектирования*.

Из статьи Мишель Куин о *General Magic* видно, что сложившийся в отрасли образ мышления попросту пренебрегает очевидными выводами. Автор статьи склоняется в пользу того, чтобы возложить вину за провал продукта на высокомерие и эгоизм Пората, только в Кремниевой долине не найти президента компании, которому не были бы присущи эти качества в непомерном количестве. Высокомерие и эгоизм уж точно не могут привести к провалу компании.

Культура высоких технологий так прочно въелась в наше общество, что мы едва ли замечаем свои провалы и слабости. Нельзя стать успешным журналистом в сфере высоких технологий, если вы не являетесь подкованным в компьютерах умником – апологетом, потому журналисты склонны считать причиной наших провалов наших собственных демонов, злой рок или волю Божью.

* * *

Разработку программного обеспечения нельзя назвать полноценной профессией, как, например, юриспруденцию, архитектуру или медицину, так что не стоит полагаться на названия должностей в нашей индустрии. Несколько моих друзей, первоклассных разработчиков, называют себя «проектировщиками программного обеспечения», что на самом деле не совсем корректно. Например, Энди Херцфельд охотно позволяет называть себя «проектировщиком».

Многие программисты склонны считать себя талантливыми проектировщиками. На самом деле обычно действительно так оно и есть, однако существует превеликая разница между тем, что называется «проектирование для *функционирования*» и «проектирование для *людей*».

Но даже если программисты не оправдывают себя в деле проектирования, они, по крайней мере, не позволяют большинству проектов окончательно развалиться. Когда «враг» ступает на их территорию, они пытаются не позволить безответственным людям захватить контроль. Многие программисты в достаточной мере ответственны и часто расценивают внешних консультантов, маркетологов и руководителей как взбалмошных и не слишком компетентных личностей.

Программисты как будто обладают специальными детекторами, различающими подвох за версту, и, чтобы эти детекторы начали активно улавливать внешнее вмешательство, маркетологам или менеджерам требуется всего пару раз заявить о «необходимости внести улучшения в интерфейс», и после этого на программистов уже ничего не действует. Какими бы ни были эти изменения – хорошими или плохими, они добавляют программистам работы. Кроме того, каждое такое изменение понижает общее качество кода, поскольку неизбежно оставляет после себя некрасивые сращивания и рубцы. Когда кто-либо заявляет, что программа станет проще в использовании, стоит только перенести все кнопки «OK» во всех диалоговых окнах в правый верхний угол, опыт и здравый смысл программиста подсказывают ему, что это просто потеря времени – *его времени*. И в общем-то его опасения вполне оправданны.

После нескольких подобных сумасбродных затей программисты начинают рассматривать все поступающие от сторонних людей указания относительно проектирования лишь в качестве рекомендаций. Они похожи на строителей, которые разрушили слишком много необдуманно возведенных стен и теперь косо смотрят в сторону планов, решив для себя, что больше никогда не воспримут их всерьез.

* * *

На маркерных досках в своих офисах разработчики программного обеспечения рисуют схемы, где бэкенд программы (код, описывающий процедуры обработки данных) и ее фронтенд (код, описывающий пользовательский интерфейс) представлены двумя разными прямоугольниками. Но на самом деле разницы в коде между ними нет. Это вовсе не то же самое,

как если бы одна стена была сложена из гранитных блоков профессиональным каменщиком, а вторая сколочена из досок плотником и обшита гипсокартоном при помощи соответствующего специалиста. Напротив, работа с переменными, указателями, вызов процедур по большей части идентичны, вне зависимости от того, происходит ли обработка клика мыши пользователя или реорганизуется база данных в глубине компьютерных недр. Нередко один и тот же разработчик пишет внутренний код системы, и он же описывает в коде пользовательское взаимодействие. Он использует один и тот же язык программирования, библиотеки, инструменты и методы при решении обеих задач. Кто рискнет точно предположить, где проходит грань между программой для компьютеров и программой для людей?

Программисты обычно распределяют задачи, коррелируя их с функциями будущего продукта, поэтому им совершенно непонятно, как может быть хорошей мысль взять отдельный фрагмент программы, нарушая множество границ функциональности, и превратить его во внешний элемент. Инженерам трудно взять в толк, почему нужно считать различными код на языке C, описывающий взаимодействие с базой данных, и аналогичный код на языке C, описывающий взаимодействие с пользователем.

* * *

Джим Гэй, мой коллега, поведал мне другую историю. В ней наглядно демонстрируется, как легко высокоинтеллектуальные инженеры увлекаются решением проблем, которые вызывают у них интерес и азарт, лишая внимания те задачи, которые на самом деле требуют решения.

Наш стартап TransPhone намеревался выйти на рынок электронной коммерции. Главной идеей было разработать легкий в применении смартфон, с помощью которого клиент мог бы выполнять операции с покупками через интернет. Важнейшей составляющей для успеха всего проекта должен был стать простой, легкий в использовании интерфейс, с которым могли бы комфортно работать даже люди, малознакомые с компьютерами. Стартап TransPhone привлек для помощи в описании такого интерфейса компанию, чьей специализацией были пользовательские взаимодействия.

По нашему мнению, мы уже фактически разработали пользовательский интерфейс, и его только требовалось слегка отладить. Тем не менее уже в первую встречу с проектировщиками этой компании они начали настойчиво твердить нам, что никак не могут понять, что мы хотим получить в итоге и кто будет пользоваться этим продуктом. Нам показалось, что они чрезмерно упрощают взгляд на проблему, которая в действительности была очень серьезной. Встреча закончилась тем, что проектировщики поручили нам обозначить наши цели более четко. Мы же со своей стороны начали подумывать о том, что у этих проектировщиков нет ни малейшего представления о том, что мы хотим получить в итоге.

Мы усовершенствовали прототип, чтобы затем показать его нашим потенциальным партнерам, однако устройство TransPhone просто-напросто не вызвало у них энтузиазма. Мы продолжали думать, что все дело было в недостаточной убедительности нашего прототипа. А спустя несколько недель после того, как был создан второй прототип, стартап TransPhone приостановил свою деятельность.

Когда я мысленно возвращаюсь к той встрече с проектировщиками взаимодействия, я теперь ясно понимаю, что они выявили нашу главную проблему уже в первые несколько минут беседы, и заключалась она в следующем: какие цели мы преследовали и для кого мы все это пытались создать? На эти вопросы они так и не добились от нас адекватного ответа. Если бы мы задали себе эти вопросы еще в самом начале, то, вероятно, произошло бы одно из двух: мы или нашли бы ответ на них (и обрели надежду на успех), или не смогли бы ответить (и тем самым минимизировали потери средств инвестора).

Мораль этой истории такова: дизайн и проектирование – это решающая часть жизненного цикла продукта. Наше нежелание обратиться к базовым вопросам проектирования и предпочтение отдать приоритет техническим аспектам разработки и продажам в итоге обрекли нашу компанию на верную гибель. Окидывая взглядом прошлое, я понимаю, что в тот момент, когда мы осознали, что не можем понять, чем мы на самом деле занимались, нам следовало пересмотреть первоначальные гипотезы нашего предприятия. Полагаю, это в конечном счете

привело нас к созданию другого, более простого продукта. Мы же добавляли все новые и новые «примочки», что, вероятно, сделало наше ценностное предложение еще более размытым.

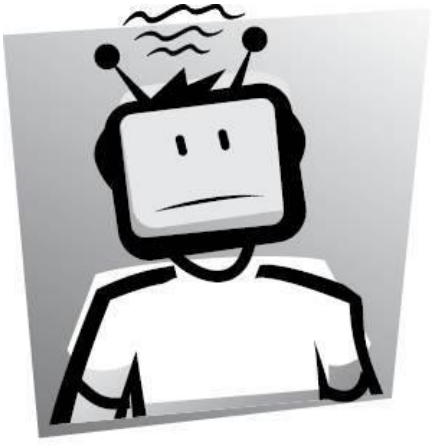
Печальный опыт Джима, как и опыт ребят из *General Magic*, показал, что прорывной технологии и разогретого рынка недостаточно для преодоления силы тяжести плохо продуманного кода. Недостаточно лишь возвести мост между технологией и потребностями пользователей. Нужно еще побудить людей этот мост перейти.

Исторически развитие технологий уходит корнями в индустриальную эпоху, потому проблемы и решения, которыми оно характеризуется, зависят от развития человека. Между людьми и механическими устройствами существует определенное сопротивление, но также наблюдается и баланс. С приходом эры информации, когда компьютеры заполонили нашу жизнь и все больше и больше продуктов обладают кремниевыми микрочипами, мы осознаем, что между людьми и устройствами возникает когнитивное сопротивление – нечто абсолютно для нас незнакомое, к чему мы еще так плохо подготовлены. Наши инженерные навыки отточены до совершенства, но в попытках применить их к решению проблем когнитивного сопротивления мы терпим крах. С годами наши разработчики программного обеспечения стали лучше и опытнее в профессиональном плане, тем не менее их результат в создании более мощного и приятного ПО остается на том же невысоком уровне, что и прежде.

По моему твердому убеждению, наша неудача в попытках решить проблему инженерными методами лишь доказывает, что этот способ неприемлем. Позволю себе пойти дальше и предположить, что эти самые инженерные методы и являются одной из коренных причин возникающих проблем. Требовать от инженеров исправить проблему – это все равно что требовать от лисы обеспечить безопасность курятника.

7. Homo logicus

С некоторой долей иронии я зову программистов *Хомо логикус* (*Homo logicus*): как особый вид, который совсем чуть-чуть, но довольно ощутимо отличается от вида *Хомо сапиенс* (*Homo sapiens*), человека разумного. На основании моих личных наблюдений я выделил четыре базовых способа, какими мыслят и действуют разработчики программного обеспечения и которые отличают их от обычных людей, – именно этим отличиям будет в подробностях посвящена данная глава. Программисты готовы с радостью поступиться простотой приложения ради возможности все контролировать. Они променяют успех на понимание. Они фокусируются на всех ситуациях, возможных в реальности, вместо того чтобы учитывать только то, что более вероятно. А еще они ведут себя довольно жестоко.



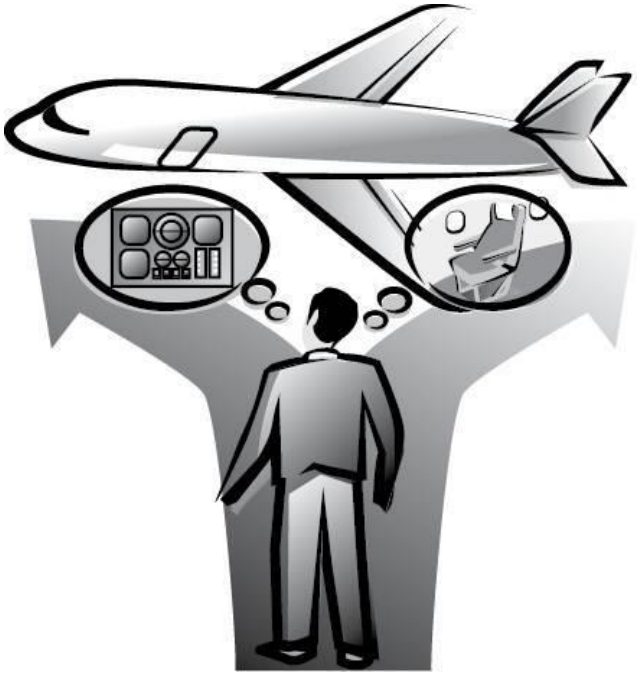
Homo logicus

Самолетный тест

Одна маленькая забавная проверка, которую я называю «самолетный тест», помогает мне безошибочно отличить один вид от другого. Чтобы посмотреть, как она работает, достаточно проделать следующее: представьте, что проходите по посадочному коридору к самолету. Когда вы ступаете на борт, вам нужно выбрать, куда пойти: налево – в кабину для пилотов или направо – в салон для пассажиров.

В кабине для пилотов вас ожидает целый калейдоскоп сложных управляющих элементов и счетчиков на приборной панели – везде, куда ни глянь, расположились всевозможные индикаторы, тумблеры и рычаги. Справа – резкий контраст: пассажирский салон с мягкими обтекаемыми линиями, в умиротворяющем бежевом цвете.

Повернете налево – и вам придется мастерски овладеть всем этим сложным техническим хозяйством. Знать назначение каждого из приборов. В обмен на понимание всех этих инструментов вы получите ощущение, что *все контролируете*, что вы надежны и способны посадить самолет в нужном месте.



Повернете направо в салон – значит, снимете с себя всякую ответственность за полет. Отказавшись от контроля, взамен вы получите возможность отдохнуть и расслабиться, осознавая, что попадете в нужное место без необходимости разбираться в чем-то более сложном, чем включение и выключение лампы для чтения.

Самолетный тест позволяет четко отнести человека к одной из двух категорий: тот, кто предпочел бы свернуть налево, обладает явным желанием держать все под контролем и понимать, как работают технические приспособления, а тот, кто свернул бы направо, обладает явным желанием упростить процесс размышлений и иметь уверенность в том, что полет завершится успешно. Программистам – представителям *Хомо логикус* – всегда хочется свернуть налево. А обычным пользователям – представителям *Хомо сапиенс* – всегда хочется свернуть направо.

Как устроена психология программистов

Казалось бы, раз нашей целью является создание программных продуктов, функциональных и приятных пользователю, то естественным желанием будет хотеть понимать психологию пользователя. Безусловно, это справедливо, однако здесь скрывается не столь явная, но очень важная мысль. Найти решение проблемы и претворить это решение в жизнь – это совершенно разные вещи. Для меня было бы более предпочтительно получить лишь частично продуманный, но уже реализованный продукт, чем блестящий дизайн-проект, величественно возлежащий на пыльной полке. Чтобы проекты наших программных продуктов были реализованы и дошли до конечного пользователя, нужно выполнить еще одно важное условие: познать психологию тех, кто этот продукт создает, – программистов.

Никаких изменений не произойдет до тех пор, пока мы не сможем повлиять на разработчиков программного обеспечения. Даже если бы программисты согласились с тем, что с

пользователями следует лучше обращаться – а чаще они все же соглашаются, – из этого не следует, что они в самом деле сделают все возможное, чтобы достичь этой цели. Недостаточно просто попросить их измениться. Чтобы решить проблему эффективно, потребуется приобщиться к образу их мышления и понять, как замотивировать их создавать интерактивные продукты, приятные пользователям. Проектировщику взаимодействия важно обладать знаниями психологии, но сюда должно входить не только понимание психологии пользователя, но и психологии разработчиков программных продуктов.

Понять назначение этого очень просто: программисты не похожи на обычных людей. Отличия в стереотипах их поведения уже давно стали объектом насмешек: неуклюжее общение с другими, защитные вкладыши для карманов[16], оторванность от жизни. Это лишь то, что лежит на поверхности, – такие отличия заметны и подвержены высмеиванию. Настоящие же отличия не просто менее различимы, но именно они имеют такое сильное влияние на те взаимодействия, которые разрабатывают программисты и которые обладают высоким уровнем когнитивного сопротивления.

Выявление этих отличий стоило громадных усилий многим обозревателям компьютерной индустрии. Так, Роберт Крингели называет программистов «зловонными божествами в мире простых смертных», подразумевая как их высокомерие к другим людям, так и сложные отношения с личной гигиеной.

По Бронсон – еще один наблюдательный обозреватель и талантливый писатель, который направлял свой проницательный взор и еще более проницательный ум на мир высоких технологий. Передразнивая Стивена Кови, он написал список «Семь навыков высокоэффективных инженеров». Пункты этого списка невероятно точно отражают суть, хотя и слегка преувеличены.

1. Они безгранично великодушны в своем эгоизме.
2. Чем меньше они видят, тем лучше это для них.
3. Они укусят не только руку, их кормящую, но и собственную руку.
4. Они приложат максимум усилий, чтобы поддерживать свой имидж в таком состоянии, чтобы все думали, что их это не заботит.
5. Они будут чинить и чинить то, что не сломано, пока оно окончательно не сломается.
6. «Это не я неверно ответил, это вы не так спросили».
7. Считают отсутствие критики лучшей похвалой.

Программисты поступаются простотой в угоду контролю

Человек вида *Хомо логикус* хочет получить контроль над тем, что представляет для него интерес, а к этому относятся детерминированные системы, характеризующиеся особой сложностью. Люди – существа, обладающие такой сложностью, однако их поведение, в отличие от поведения механизмов, не поддается логике и трудно предугадывается. Лучшими механизмами программист считает цифровые механизмы, потому что они наиболее сложны, запутанны и их можно легко модифицировать.

Контролировать людей является занятием менее привлекательным, на взгляд программиста. По Бронсон в своем романе *The First $20 Million Is Always the Hardest*[17] («Первые двадцать миллионов долларов самые трудные») описывает, как побуждал программистов разыгрывать людей в целях показать им, что они могут контролировать их, но этот эксперимент показал, что программисты получают большее удовольствие, когда заставляют компьютеры подчиняться их власти.

Расплата за возможность все контролировать всегда приходит в виде дополнительных усилий и увеличивающейся сложности. Большинство людей готовы прилагать умеренные усилия для выполнения задач, но программистов в этом отношении отличает желание и способность мастерски овладевать системами с невероятной сложностью. Одно из самых приятных дел в работе программиста – управлять системами, состоящими из множества взаимодействующих компонентов. Пилотировать самолет – вот истинное призвание программиста[18]. Приборная доска самолета пестрит множеством всяческих датчиков, рычагов и тумблеров, но программисты справляются с этими сложными элементами вполне успешно. Представители вида *Хомо логикус* считают такое занятие весьма забавным времяпровождением, несмотря на то (а точнее

сказать – благодаря тому!), что скрупулезно осваивать управление приходится долгие месяцы. Человек вида *Хомо сапиенс*, напротив, выбирает полет в пассажирском салоне.

Для представителей *Хомо логикус* контроль является главной целью, а сопутствующая ему сложность – просто плата за достижение цели. Для обычных людей целью является простота, а платой за это, которую они готовы отдать, – отказ от возможности все контролировать. В программных продуктах возможность контроля выражается в различных функциях и опциях. Например, в операционной системе Windows 95 функция «Поиск файла» предоставляет мне возможность контролировать процесс. Я могу выбирать расположение на диске для осуществления поиска, указывать тип файлов, которые нужно найти, уточнять, нужно ли искать по именам файлов или по их содержимому, и другие параметры. Если смотреть на это взглядом разработчика, то это просто замечательно. Потратив чуть больше усилий и получив чуть более глубокое понимание сути вещей, он расширяет свои возможности и может выполнять поиск с большей скоростью и эффективностью. А вот со стороны пользователя все представляется не так радужно, ведь он вынужден указывать все нужные параметры, вроде места для поиска, типа файла и способа поиска по имени или по содержанию файла.

Люди вида *Хомо сапиенс* с радостью пожертвовали бы каждой лишней минутой времени, проведенным за компьютером, если бы не были вынуждены разбираться, как устроена функция поиска. С их точки зрения, каждый дополнительный параметр поиска – это еще один риск сделать что-нибудь не так. Чем более гибкой становится система, тем больше возрастает вероятность допустить ошибку и получить некорректные результаты поиска, хотя поначалу это кажется не так. Простые пользователи с радостью пожертвовали бы всей этой бесполезной сложностью, возможностью все контролировать и разбираться в предмете, лишь бы их работа была проще.

Программисты готовы променять успех на понимание

Видом *Хомо логикус* движет неистощимое стремление к познанию того, как работают вещи. В противоположность ему вид *Хомо сапиенс* направляет все свои усилия на то, чтобы добиться успеха в чем-либо. Программистам тоже свойственно желание преуспеть, однако они готовы принять неудачу, сочтя это ценой за понимание.

Один старинный анекдот про инженеров поможет чуть глубже разобраться с тем, что подразумевается под пониманием:

Встречаются как-то три приговоренных к казни человека: священник, адвокат и инженер. Первым на эшафот ведут священника. Палач тянет рычаг, отпускающий нож гильотины, но тот не опускается. Священник восклицает, что это божественное вмешательство, и требует освобождения, так что его выпускают на свободу. Следующим выходит адвокат. Палач снова

тянет рычаг, и вновь ничего не происходит. Адвокат заявляет, что еще одна попытка будет считаться вторичным привлечением к ответственности за то же преступление, требует отпустить его и выходит на свободу. Наконец приходит черед инженера, он поднимается на эшафот и начинает тщательно изучать устройство гильотины. Палач тянет рычаг, и, прежде чем лезвие опускается на шею инженера, он успевает взглянуть вверх и воскликнуть: «А, так вот в чем была проблема!»

Суть этого анекдота в том, что для инженера понять механизм работы гильотины было важнее, чем сохранить собственную жизнь.

Когда я читаю лекции программистам, я всегда прошу поднять руки тех, кто в детстве разбирал часы, чтобы посмотреть, как они устроены. Обычно руки поднимают как минимум две трети присутствующих в аудитории. Следом я задаю такой вопрос: «А скольким из вас в итоге удалось собрать часы обратно?» – после этого опускается большая часть рук. Далее я спрашиваю, кто их них считает это неудачей? Почти все смеются, осознавая, что получают удовольствие, когда потрошат свои часы. Вид *Хомо логикус* стремится разобраться, как устроен часовой механизм, – это его конечная цель, и он легко готов пожертвовать работающими часами в обмен на понимание их сути. Его противоположность – *Хомо сапиенс* – любит, когда часы работают. Он стремится к тому, чтобы получить возможность знать, который час, и при этом он готов жертвовать знанием об их внутреннем устройстве.

Джонатан Корман, проектировщик взаимодействия, говорит так:

Многие люди даже не осознают, как сильно компьютеры способны увлечь программистов. Чем сложнее устройство в изучении – тем больше удовольствия получает программист. Этот интерес обладает такой силой и глубиной, что у них не возникает и мысли, что кто-то может думать иначе, оттого они принимают за причину недовольства пользователей их нежелание тратить больше усилий на обучение работе с системой, а вовсе не недостаток интереса.

Стремление программистов понимать суть вещей побуждает их на инстинктивном уровне создавать такие пользовательские взаимодействия, которые четко соотносятся с внутренним функционированием продукта. Забывая о том, что программа должна помогать пользователю достигать его целей, они делают программы, отражающие механизм их устройства. Разумеется, сами программисты не испытывают неудобств при обращении с такими программами, так как понимают, как она устроена, и, следовательно, знают, как решать задачи с ее помощью. Мы зовем такой повсеместно распространенный способ проектирования взаимодействий *моделью реализации*. Например, документы компьютера имеют постоянное место хранения в виде различных дисков, но программам доступно изменять эти документы, только пока они временно находятся в оперативной памяти. Программисты могут с уверенностью отличить одно от

другого, поэтому в интерфейсах их программ присутствуют оба типа памяти компьютера. Пользователю такие нюансы кажутся совершенно ненужными – это все равно что разместить на приборной панели автомобиля переключатель, чтобы водитель был вынужден выбирать между диагональными и радиальными шинами.

Обычным людям вполне свойственно не хотеть разбираться в устройстве и функционировании вещей, даже если зависят от них в повседневной жизни и используют постоянно. Им кажется, что интерфейсы, созданные по модели реализации, вынуждают их к ненужному пониманию сути. Программисты не могут взять в толк, как можно думать подобным образом.

Программисты фокусируются на исключениях

Программистам присуще рассматривать сложные комплексные системы так, как это делают математики, оттого совсем не удивительно, что они не похожи на большинство других людей. Вот что я под этим подразумеваю: вообразите, что вы подкинули монетку 1 000 000 раз, и 999 000 раз она упала орлом вверх и всего один раз – решкой. С точки зрения математика, это опровергает утверждение о том, что монетка всегда падает орлом вверх. Тот единственный раз, когда она упала решкой вверх, делает все утверждение неверным. В терминах математики утверждение является истинным, только когда оно истинно *всегда*. Такой ход мыслей привычен для *Хомо логикус* и кажется им логичным, потому что именно такое поведение свойственно компьютерам.

Большая же часть обычных людей, напротив, будет считать, что описанное выше утверждение является истинным, ведь это подтверждается преобладающим количеством раз, когда монета упала орлом вверх. Более того, они также будут уверены, что это утверждение не просто истинно, но его истинность является сокрушительной, убедительной и неоспоримой. Миллион против одного, что оно верно! Свойством человеческого разума является считать подобные события, где вероятность миллион к одному, однозначными. Это вероятность, с которой не поспоришь. Меня скорее ударит молния или я случайно упаду с моста, чем монетка упадет решкой вверх.

Вероятность того, что утверждение о монетке истинно, невероятно огромна, а человек вида *Хомо сапиенс* живет в мире вероятностных событий. Тем не менее вероятность того, что утверждение ложно, все же возможна, а программисты существуют в мире возможностей. Если какое-то событие может случиться, оно должно быть учтено. В мире программного обеспечения, где решающую роль играют точные утверждения, нельзя упускать из виду даже события с ничтожной вероятностью.

Программисты обычно называют такие маловероятные события «исключениями»[19]. Вероятность наступления таких событий ничтожно мала, однако они могут вызвать серьезный сбой в работе программы, если не предусмотреть их наступление. Несмотря на их низкую вероятность, цена неподготовленности может быть очень высока. Так что маловероятные события являются вполне обыденной реальностью для программистов. И даже тот факт, что исключение может возникнуть только один раз в 79 лет при ежедневном использовании программы, совсем не меняет дела. *Что будет, если этот единственный раз случится завтра?*

Бесспорной отличительной чертой программиста-любителя от профессионала является навязчивая одержимость тщательно предусматривать возможные исключительные ситуации и готовиться к ним, как это делают путешественники. Но такое фанатичное стремление готовиться к исключительным ситуациям влечет за собой неминуемые последствия в виде пренебрежения теми событиями, которые имеют больший процент вероятности случиться. В результате на свет появляются программы, которые обладают взаимодействием, напичканным редкими или невостребованными функциями, которые затрудняют работу с тем, что действительно важно и часто применяется. Вот одна из самых распространенных жалоб пользователей: с программой сложно работать, потому что в ней слишком много разных настроек и все смешаны в одну кучу, без какого бы то ни было разделения.

Такая щедрость в добавлении множества ненужных и нежеланных функций как следствие программистского мышления в категориях возможного – это прекрасный пример того, что По Бронсон называет «великодушие в эгоизме». Они наполняют программные функциональные возможности тем, что нужно *им самим*.

* * *

Распространенная в среде программистов шутка гласит, что в мире существует только три цифры: 0, 1 и бесконечность. В применении к компьютерному миру она действительно справедлива. Внутри двоичной вычислительной системы событие либо происходит, либо нет – соответственно, это 1 или 0. Если же какое-то событие может произойти больше одного раза, значит, оно может повториться бесконечное количество раз.

Код установки программы и завершения ее работы пишется таким образом, что может быть выполнен только один раз. При попытке программы повторно выполнить этот код может возникнуть системный сбой или как минимум могут появиться ошибки в работе программы. Фрагменты кода других частей программы пишутся таким образом, чтобы выполняться повторно. Фактически любая часть программы, которая *может* выполниться не менее двух раз без сбоев, может выполняться сколько угодно раз. С точки зрения кода и программиста вида *Хомо логикус*нет большой разницы между двумя запусками программы и двумя миллионами запусков.

Мышление обычных людей отличается от мышления программистов. Они оперируют такими цифрами, как 0 и 1, но не менее уверенно они пользуются, например, цифрами 2, 7 и числом 31. Большинству людей гораздо легче представить 300 вещей, нежели миллион. Обычный человек исчисляет вещи такими количественными величинами, которые неведомы программистскому разуму. К примеру, ярые поклонники катания на лыжах могут потратить на это занятие около десяти выходных в сезон. Если взять 40 лет активного катания, то в общем количество таких вылазок будет менее 500 – и это за всю жизнь! Для современного цифрового компьютера обработка 500 объектов займет доли секунды. Даже самый увлеченный пользователь какой-либо программы запустит ее не больше нескольких тысяч раз, при этом программисты все еще мыслят категориями бесконечных чисел.

Хорошим программистам свойственно целенаправленно закрывать глаза на существование на практике таких чисел, как 500, так как это позволяет им подготовить программы к возникновению 501-го раза. Именно это подразумевает По Бронсон под описанием второго навыка высокоэффективного инженера: «Чем меньше они видят, тем лучше для них».

Программисты ведут себя жестоко

Наверное, самая большая вещь, способная поразить в хорошем программисте, – это то, что он ведет себя как качок. Я намеренно употребляю это слово, потому что оно вызывает ассоциации с незрелостью, эгоизмом, желанием соперничать, а также с немереной силой и ловкостью.

При слове «качок» перед моими глазами предстают уроки физкультуры в средних классах. Некоторых ребят природа наделила хорошо развитыми, крепкими мускулами и ловкостью. Они преуспевали в школьных мероприятиях, требующих хорошей спортивной подготовки, но также быстро поняли, что могут довлеть над другими детьми, невысокого роста и более слабыми, в учиненных ими негласных поединках в силе и гибкости. Эти «качки» главенствовали не только на площадках для игры в бейсбол или футбол, но и в раздевалке, а также на школьном дворе, участвуя в несанкционированных стычках.

Юноша 17 лет от роду ростом 180 сантиметров силен, как взрослый мужчина, но уступает ему в зрелости. Этот полумужчина не жалует тех, кто слабее его. Подростковый период мучителен для него, но он еще не готов выносить сложности мира взрослых. Он руководствуется простой жизненной философией: защищайся или умри. Своими поступками он транслирует сообщение: «Если не можешь сделать, как я, то ты просто бесполезный неудачник». Любой парень во дворе, кто не может ему противостоять, награждается презрением и считается недостойным. Обладая несметной физической силой, позволяющей ему главенствовать, он и пытается это делать.

Со временем с этим качком происходит нечто интересное. Распрощавшись со школой, он попадает в реальный мир, где его способность главенствовать за счет своей физической силы быстро становится бесполезной, и он теряет свою власть. Случись так, что какой-нибудь круглолицый умник в очках начнет насмехаться над таким качком, пара ударов и одобрительный гогот дружков громилы восстановят его авторитет. В мире бизнеса использовать кулаки и дерзость не получится. Конференц-зал – не место для пинков и хлестанья полотенцем, там это будет просто неэффективно. Несмотря на то что качок по-прежнему физически силен и способен одолеть любого более слабого человека, окажись на месте такого слабака его коллега, супервайзер или руководитель, качку придется несладко.

Качки, которые были такими незрелыми в школьные годы, усваивают весьма уничижительный урок. Очутившись в большом мире взрослых, они чувствуют себя в обществе так, будто им обрезали крылья, им приходится учиться успешно сосуществовать с другими людьми, менее сильными, чем они. В бизнесе можно часто встретить таких «качков», и в конечном счете они весьма неплохо себя проявляют. Переход из одного состояния в другое они осуществляют успешно, хотя поначалу не готовы к этому и не хотят меняться. Внутри них все еще жив дух соперничества, но теперь они достигли того уровня зрелости и самоотдачи, который помогает им стать достойными членами общества.

Программисты очень сильно напоминают таких качков. Будучи школьниками, многие из них не обладали таким уровнем физического развития, как у настоящих качков, но они опережали своих сверстников другими качествами: природа даровала им быстрый, развитый ум и прекрасные мыслительные способности. Они преуспевали в различных школьных занятиях: аналитической и исследовательской деятельности, литературных клубах, шахматных поединках.

Если говорить о трудностях переходного возраста, то здесь развитые умственные способности программистов ценились не так высоко, как крепкие мускулы. На школьном дворе они легко подпадали под влияние более сильных товарищей. Худощавый семнадцатилетний подросток, сравнимый со взрослым познаниями в математике, физике или компьютерных науках, все еще остается физически неразвитым ребенком, которого презирают на футбольном поле и не приглашают на свидания. Таких детей обычно называют «ботаниками».

Ботаники не жалуют тех, кто уступает им в интеллектуальном плане. Они смеются и подшучивают над теми, кому недостает их интеллекта, но делают это лишь в собственных мыслях, потому что не обладают физической силой, чтобы рисковать насмехаться над кем-то у всех на виду. Жизненная философия ботаника жестока и проста: защищайся или умри. Любой другой, кто не может одолеть его в интеллектуальном плане, недостоин его внимания. Он не принимает в расчет иные способности или чувства тех, чей ум уступает его уму. Его система ценностей представляет собой простую иерархию, основанную на внутреннем развитии его умственных качеств. О тех, кто сравним с ним по интеллекту, он думает так: «Если я могу одержать над тобой победу в интеллектуальном поединке, значит, я выше и лучше тебя».

Как и качки, наделенные физической силой, программисты тоже не лишены природного таланта, равно как и обладают сильным духом соперничества, подобно молодым спортсменам. Заметить в них присутствие этого соперничества бывает непросто, поскольку программирование – игра некомандная и по большей части невидимая. Однако не стоит обманываться неприметными манерами программистов – они способы рьяно конкурировать с другими, а действительно хорошие программисты столь же беспощадны в достижении своих целей, как будущие олимпийские чемпионы.

Со временем с таким ботаником происходит нечто интересное. Распрощавшись со школой и очутившись в реальном мире взрослых, он обнаруживает, что его способность преобладать над другими в интеллектуальном плане не теряет своей силы при переходе в это общество зрелых, цивилизованных, взрослых людей. Ботаник находится под защитой социальных норм, где его уже не так легко поколотить, как на школьном дворе. Преобладание на уровне физической силы перестает быть приемлемым поведением для зрелого человека во взрослом обществе, в то время как преобладание на интеллектуальном уровне лишь становится все более сильным преимуществом.

Этот интеллектуальный качок, способный главенствовать над другими за счет своих умственных способностей, обретает невероятную мощь в эпоху информационных технологий. В современном гражданском обществе становится совершенно допустимым интеллектуально «пинать» других с помощью непонятного программного обеспечения или «хлестать умственным полотенцем» тех несчастных, кто всего лишь пытается добыть свои деньги из банкомата.

Качки, которые главенствовали на школьном дворе, теперь осознали, что очутились на растерзании у своих бывших жертв. В ходе уничижительного взросления многие из них превратились в весьма достойных людей и даже как-то признавались в разговоре со мной, что немало раскаиваются в том, как вели себя в подростковом возрасте.

По словам одного бывшего разыгрывающего защитника из сборной по бейсболу, парня довольно высокого роста, – его физическая сила не приносит никакой пользы в зале заседаний, а вот бывшему казначею школьного астрономического клуба, который ростом особо не выделялся, его интеллектуальная сила позволяет быть гибким и с непревзойденной меткостью поражать противника. Адвокат – в прошлом незрелый «ботаник» – будет блистать в суде, поражая всех своим острым языком и непревзойденным умом. Доктор с таким же прошлым теперь держит в руках жизни своих пациентов-качков. А ботаник-программист – сюрприз! – оказывается наделен самой мощной силой из всех когда-либо ему доступных – он контролирует доступ всех людей к жизненно важной информации.

Теперь никакой процесс взросления не сдерживает проявление этой силы. Они властвуют над другими за счет своих интеллектуальных способностей, потому что они это могут, и не видят ничего предосудительного в том, чтобы издеваться над пользователями посредством пугающе сложных программных продуктов. Они отпускают презрительные насмешки и шутки в адрес «чайников», которым недостает ума, чтобы пользоваться компьютером. Кроме того, их привычка работать в одиночестве, напряженно и сверхурочно, отнюдь не прибавляет им навыков социализации.

Только лишь почти в тридцатилетнем возрасте я осознал, как грубо вел себя. От настоящих качков меня отличало лишь то, что моими кулаками были способности к программированию, а мастерское владение сложными системами заменяли мне высокий рост и длинные руки. И я так по-свински обращался с теми, кто был не в силах одолеть сложности при использовании компьютера.

8. Вымирающая культура

Программирование – в некоторой мере чуждая человеческому разуму деятельность, но вместе с тем она способна очень сильно воздействовать на эмоции. Благодаря именно этому свойству программирование можно считать призванием. Язык, на котором общаются разработчики, похож на особый диалект, присущий всему братству программистов с собственной сформировавшейся культурой. В этой главе я продемонстрирую, что происходит с сущностью программных продуктов под влиянием этой культуры программирования.

Культура программирования

Как-то в воскресном выпуске газеты я прочел одну занимательную историю об американской супружеской чете, которая переехала в Мексику после выхода на пенсию. Они приобрели земельный участок в пригороде мегаполиса и пригласили архитектора из Америки спроектировать дом их мечты. После этого они передали планы для постройки мексиканскому подрядчику. К их величайшему удивлению, в ходе строительства стало заметно, что дом обретает совсем не те очертания, какие присутствовали на планах архитектора.

На плане значилось, что вдоль фасада дома должно быть установлено четыре окна, производитель и номер модели которых были четко обозначены. Взору владельцев дома же явилось только три окна, крайне отличных от нужных не только по марке производителя, но и по размерам и внешнему виду. Когда они указали на это мексиканскому строителю, тот лишь пожал плечами и ответил: «Это окна. На плане значится, что окна должны быть на этой стороне. Что не так?»

Владельцы дома и архитектор были представителями одной культуры и обладали одними и теми же ценностями, в то время как строительный подрядчик представлял другую культуру, а потому видел проблему под другим углом. Конечно, ему удалось закупить окна за меньшие деньги при меньшем количестве усилий, а в привычном ему мире эти аспекты выходили на первый план. В свою очередь, американский архитектор и сами владельцы считали, что при наличии плана ему нужно следовать четко. Мексиканский строитель же был уверен, что планы являются лишь приблизительными рекомендациями, а не жестким требованием. Он думал, что его личные представления об экономии и легкости приобретения вполне естественно превалируют даже над самыми точными спецификациями. Он с чистым сердцем пытался реализовать видение архитектора, но пропускал все через свои внутренние культурные фильтры – собственную оценку ситуации.

Повторное использование кода

Как и тот мексиканский строитель, ценивший стоимость проекта выше его дизайна, так и программисты, предоставленные сами себе, поставят эффективность программирования выше, чем потребности пользователя. Лучше всего это утверждение доказывает практика повторного использования кода, написанного ранее для какого-либо другого проекта или приобретенного за символическую цену у сторонних разработчиков. Готовый код не просто экономит время – он уже доказал свою пригодность у других программистов, а кроме того, в нем определенно отсутствуют баги. Для программ характерно одно уникальное свойство – любую процедуру в них можно вызвать с помощью всего лишь одной команды, при этом размер процедуры значения не имеет. Получается, что, если процедура уже написана, требуется лишь одна строка кода, чтобы осуществить ее вызов. Таким образом, любой готовый программный модуль представляет собой отличное подспорье для разработчиков. Они могут подключать такие блоки кода к своим программам, рассматривая их как черные ящики, в работу которых нет необходимости вмешиваться. Такой подход влечет значительную экономию не только во времени программирования, но и во времени обдумывания задачи и ее тестирования. Большинство программистов придают повторному использованию кода невероятную важность – выше, чем любому другому техническому соображению. Идеолог открытого программного обеспечения Эрик Рэймонд как-то сказал: «Хороший программист знает, как писать код. Великий программист знает, где взять готовый».

Главным побочным эффектом повторного использования кода является то, что значительные куски большинства программ существуют вовсе не потому, что этого хотел какой-нибудь проектировщик взаимодействия, а по той причине, что какой-то программист уже написал их за чей-то чужой счет. Многие программы из тех, с чем нам приходилось взаимодействовать, существуют по той единственной причине, что уже когда-то ранее были написаны.

Вот пример: наши настольные приложения состоят из такого большого количества меню и диалоговых окон с вводом текста из-за того, что во всех операционных системах с оконным интерфейсом – Microsoft Windows, Mac OS, OS/2, Linux – имеются готовые блоки кода для обеспечения функционирования этих компонентов. И напротив, ни в одной из этих систем не существует достаточного количества готового кода для работы с элементами посредством операций drag-and-drop, оттого в приложениях так редко используется технология непосредственного манипулирования (direct manipulation). Диалог создается за 6–8 строк простого декларативного кода, в то время как для описания механизма drag-and-drop требуется около 100 строк весьма замысловатого процедурного кода. Для программиста выбор очевиден. Выгода для конечного пользователя в погоне за этой экономией как-то упускается.

Я постоянно наблюдаю, как снова и снова в разработке программного обеспечения возникает эта история с мексиканским строителем; по большей части это происходит из-за того, что программисты вынуждены прибегать к повторному использованию кода. Так, Эд Форман, руководитель разработки программного обеспечения в компании *Elemental Software*, создает детализированный и точный набросок того, что он хотел бы видеть на экране, и делает это до

того, как отдать задачу в работу программистам. И все равно, по словам Эда, программа, которую он получает в итоге, всегда представляет собой лишь блеклое подобие того, что он набросал.

Вот как это происходит: в наброске Эда кнопки темно-серого цвета размещены на фоне светло-серого цвета. Программист начинает свою работу с того, что копирует исходный код из какой-либо уже работающей части программы. Таким способом можно хорошо сэкономить время и усилия программиста, что кажется выгодным для всех – не считая того, что в готовом коде кнопки обведены незапланированной рамкой темно-серого цвета. Вместе с темно-серой рамкой добавляется и текстовая надпись. Программист мог бы удалить текст и рамку, так как в наброске Эда их нет, но он предпочитает оставить все как есть, тем самым сохраняя много строк кода. В коде заложено, что каждая текстовая надпись должна быть заполнена, потому программист просто впечатывает нечто подходящее – с его технической точки зрения.

Когда Эд видит получившийся результат с неизвестно откуда взявшимися рамками и сбивающими с толку текстовыми надписями, он удивленно качает головой. Когда затем он обращает внимание программиста на эти отличия, тот просто не видит, в чем суть проблемы. Программисты, подобно тому мексиканскому строителю, считают свои убеждения относительно простоты создания программ и легкой добычи готового кода более важными, чем любые *рекомендации*, предложенные кем бы то ни было.

Эд не только удивлен, но и ужасно расстроен этим явлением, но объяснить, где кроется причина, он не в состоянии. Все его программисты без исключения умны, обладают недюжинными способностями и всерьез беспокоятся о качестве продукта и успехе компании, но тем не менее легко попадаются на зов сладкоголосых сирен. Конечно, они стараются претворить в жизнь видение Эда, однако не готовы при этом поступиться собственными понятиями в области реализации программ.

* * *

Особенно потрясают в привычке программистов использовать готовый код их попытки адаптировать под свои нужды даже код сомнительного происхождения. Стоит первой попавшейся идее по проектированию взаимодействия взбрести в голову программиста, как он хватается за нее при реализации, и на основе такого однажды написанного кода строятся все последующие программы, благодаря агрессивному повторному использованию кода.

Например, в операционной системе Windows ядро было написано очень опытными программистами, а самые первые тестовые приложения, целью которых было показать приемы взаимодействия с пользователем сторонним разработчикам, писали плеяды практикантов и младших программистов самой компании *Microsoft*. Внутренний код ядра Windows прошел через шесть главных релизов – его непрестанно дополняли и переписывали, в результате чего он неуклонно совершенствовался. А вот внутри подавляюще большого количества общеизвестных приложений до сих пор сохранились длинные пассажи кода, написанные двадцатилетними студентами, прибывшими на летнюю практику в Редмонд. Аналогичная ситуация наблюдается со Всемирной паутиной. Дилетанты, любящие поэкспериментировать, «состряпали» первые веб-сайты, а те, кто пришел после них, просто-напросто клонировали эти сайты, которые, в свою очередь, были позже склонированы другими.

Как вы могли увидеть, наблюдается явный конфликт интересов между потребностями пользователей и нуждами программистов. Предчувствуя такие конфликты в бесчисленном множестве профессий и занятий, мы придумываем защитные механизмы, призванные сдержать пагубное влияние конфликта. У судей и адвокатов схожие профессиональные компетенции, однако мы никогда не позволим адвокату занять место судьи в собственном судебном разбирательстве. Как не позволим и баскетболисту быть рефери в собственной игре. Конфликт интересов очевиден, тем не менее мы постоянно позволяем программистам принимать решения по проектированию на основе лишь собственных убеждений в области реализации.

В сфере разработки ПО, равно как и в корпоративных ИТ-департаментах, сложилось мнение, что никто так не подходит для проектирования программ, как программисты, ведь они являются узкими специалистами с глубоким пониманием нюансов в упомянутых областях. И хотя нам кажется вполне невинным и естественным занятием дозволить программистам самим выбирать

форму и поведение тех программ, которые они создают, ловушки конфликта интересов в этом случае не избежать. И ловушка эта коварна не столько различиями между программистом и пользователем, сколько их схожестью. Пользователь преследует свои цели, а программист – свои. Суть проблемы заключается в едва уловимой разнице между целями обоих.

* * *

Практика использования повторного кода настолько плотно въелась в сущность программистов, что они нередко прибегают к уже привычным методам, даже когда фактически не копируют сам код. Это происходит само собой, подкрепляясь склонностью программистов к консервативному поведению. Например, значительная часть программ содержит множество диалогов подтверждения, хотя на самом деле нужда в них отсутствует. Многие из них остались там, потому что изначально присутствовали в повторно используемом коде, но еще большая часть существует там потому, что программисты просто включают их в код по привычке.

Как-то на одной из конференций я столкнулся с Джеффом Безосом, основателем *Amazon.com*, и рассказал ему, как восхищен «интерфейсом в один клик» на этом сайте. Благодаря такому интерфейсу потребитель может заказать любой товар – сюрприз! – в один клик мыши. И это действительно здорово, потому что все ненужные элементы убраны из интерфейса, что позволяет пользователю нажать всего лишь одну кнопку, без всякой необходимости повторно вводить информацию по доставке и платежные данные.

Джеффри был доволен тем, что я оценил «интерфейс в один клик», и он поведал, как задумал эту идею вместе со своими проектировщиками, а потом показал программистам, которые, как обычно, покивали и заверили, что смогут это реализовать. Спустя некоторое время программисты представили результаты Джеффу. Он выбрал книгу из каталога товаров на сайте и сделал тот единственный клик по кнопке обработки заказа, после чего программа вывела экран подтверждения. Программисты просто-напросто сделали из интерфейса «в один клик» интерфейс «в два клика». С точки зрения программистов, добавился всего лишь один щелчок мышью – невелика беда. Но с точки зрения Джеффа и любого другого пользователя, это было равнозначно стопроцентному росту инфляции. Джеффу пришлось задействовать метод кнута и пряника, пока программисты не создали «интерфейс в один клик», который действительно позволял купить товар за один щелчок мыши. Конечно, Джефф не рассказал мне, насколько выросли продажи с переходом на «интерфейс в один клик», но я могу с уверенностью утверждать, что стал покупать книги на Amazon в два раза чаще.

Подобное поведение программистов мне довелось наблюдать бесчисленное количество раз, это происходило даже с самими сознательными и способными из них. Они получают от нас кропотливо созданные прототипы экранов и видят в них только смутные рекомендации к тому, каким может быть интерфейс программы. Они получают от нас списки опций и функций и выбирают из них лишь те, что отвечают их собственным предпочтениям и которые легко реализовать.

Общая культура

Природа военных действий и требования армейской подготовки схожи во всех странах. Это приводит к возникновению сильнейшего сходства в культурах всех солдат, вне зависимости от того, на чьей они стороне. То же самое происходит и в индустрии компаний, занимающихся разработкой программного обеспечения.

Разработчики программ, принадлежащие к виду *Хомо логикус*, объединены общей культурой, порожденной их коллективной психологией. Тот общепризнанный способ, каким создаются программные продукты, на удивление одинаков для компаний, выпускающих камеры, для автокомпаний, банков и морфлота, потому такие, казалось бы, разные продукты, как фотоаппарат, автомобили Porsche, банкоматы и крейсеры, оснащенные системой «Иджис», все как один имеют такое узнаваемое, схожее с компьютерами поведение.

Благоговение перед технической подкованностью – также одно из характерных для этой культуры явлений. В результате существования этого явления возникает другой эффект – важность владения техническими навыками проецируется на другие сферы, даже на те, где в таком поклонении нет необходимости, например на проектирование взаимодействия. Тридцать

лет назад, когда компьютеры размещали в специальных серверных с окнами (glass houses), к которым имели доступ только специально обученные программисты, проектирование на основании собственных предпочтений разработчиков имело смысл и отвечало требованиям времени. Когда впоследствии компьютеры постепенно стали выходить на потребительский рынок, программисты все еще занимались проектированием, потому что так исторически сложилось. Руководители разработки задаются вопросом: «Зачем мне платить проектировщикам взаимодействия, если я и так уже получаю эту работу от программистов, притом бесплатно?» Вопрос справедлив, только в его основе лежит изначально неверный довод. У этого руководителя не будет никакого проектирования взаимодействия – ни платно, ни бесплатно. Все, что он получит в итоге, – это интерфейс, спроектированный лишь для удовольствия его создателей: людей с особой подготовкой, собственной индивидуальностью и нетипичными склонностями.

Здесь мы можем увидеть еще одну ключевую особенность культуры разработки программ. Несмотря на то что эта культура базируется на специфической природе программистов, их руководители также оказывают на нее существенное влияние, потому что сами, надо сказать, когда-то были программистами. Джефф Безос упомянул, что громче всех «интерфейс в два клика» защищал менеджер по продукту!

Благоговение перед техническими навыками влечет еще один эффект. Многие люди полагают, что для программирования нужно быть более технически подкованным, чем для проектирования взаимодействия. Эту мысль оспаривать не буду, однако я решительно возражаю против типичного следствия из этого утверждения, которое гласит, что этап программирования должен предшествовать этапу проектирования в процессе разработки. Такой подход приводит к тому, что пользователь вынужден приспосабливаться к технологии. При обратном подходе, когда проектирование взаимодействия предшествует программированию, уже технология будет подстраиваться под цели пользователя. От руководителей в индустрии высоких технологий мне доводилось слышать такую фразу: «Мы подключим к задаче проектировщиков, как только программисты доделают функционал». При таком подходе шансы проектировщика как-то повлиять на ситуацию весьма сомнительны.

Культура программирования в компании Microsoft

Силу и влияние культуры разработки ПО переоценить трудно. В 1995 году Фред Муди написал книгу *I Sing the Body Electronic*[20] («Электронное тело пою»), посвященную *Microsoft*. Базируясь на исследовании этой типичной компании по разработке программного обеспечения, книга описывает, как глубоко укоренилась в нашем обществе культура «ботаников». Фред Муди – путешествующий писатель и журналист, обозревающий компьютерные темы, провел год в стенах *Microsoft*, наблюдая за разработкой мультимедийного инновационного продукта, который впоследствии получил название Explorapedia. Муди был предоставлен неограниченный доступ ко всему, что происходило в *Microsoft*, и его книга рисует нашему взору показательную картину той жизни и культуры, которая существовала внутри ведущей компании в индустрии. Как видно по продуктам *Microsoft*, программирование глубоко почитается компанией, а вот необходимость в проектировании взаимодействия ею совсем не осознается. Книга представляет собой увлекательное исследование всего, что происходит в культуре программирования.

Вступление книги создает предпосылки для дальнейшего рассказа:

Рабочие процессы внутри *Microsoft* построены следующим образом: вокруг определенных продуктов формируются небольшие команды, предоставленные сами себе – они могут самоорганизовываться и выполнять задачи как сочтут нужным. Такой подход весьма рискован, поскольку такие команды становятся настолько неконтролируемы, что это выходит за рамки понимания типичных американских организаций.

Microsoft знаменита тем, что привлекает в свою команду невероятно талантливых, крайне напористых молодых людей практически со школьной скамьи. Вот как описывает это Муди: «Создавалось такое впечатление, будто шайка подростков пробралась в офисы какой-то корпорации, когда все ушли, и обосновалась в зале заседаний, намереваясь поиграть в бизнесменов». Еще одна примечательная черта *Microsoft* – это способность безжалостно

эксплуатировать эти юные дарования, в целях использовать их таланты по максимуму. Муди пишет: «В кампусе царит крайне суматошная атмосфера, все беспрестанно что-то выдумывают».

Эта книга являет собой удивительную летопись того, что методы работы *Microsoft* нередко бывают спорны, непрофессиональны и оказывают деморализующее воздействие. Увиденное там озадачило и самого Муди, вместе с тем он был уверен, что стал свидетелем чего-то невероятно важного. Что ему сразу бросилось в глаза – так это программисты, правящие бал. И даже в те моменты, когда они не делают этого явно, они все равно влияют на все косвенно, силой своей воли. Муди ни разу не ставит под сомнение свое или чье-либо другое убеждение, следует ли программистам и *в самом деле* быть у руля, однако он постоянно упоминает сопротивление, разногласия, неприязнь и чувство неудовлетворенности, которые этому сопутствуют:

Не то чтобы я очень хорошо понимал все происходящее в *Microsoft*. К весьма безрадостным фактам я отношу то, что по выходе из кампуса компании я чувствовал себя более обескураженным, чем в самом начале. Окидывая взглядом все произошедшее, я прихожу в еще большее замешательство. Я все еще никак не могу взять в толк, как трактовать эту историю – как историю успеха, или как историю провала, или же это история успеха, скрытая за неудачей, а может быть, и история неудачи, замаскированной под успех.

Очевидно, что Фреду довелось стать свидетелем создания пляшущего медведя: тоскливого и сложного в использовании продукта, единственным лучом света в котором были опции, недоступные в каких-либо других продуктах.

Продукт Explorapedia можно назвать классическим примером того, насколько деградировал нормальный процесс разработки. Я не сомневался, что проект оказался провальным. А вот Муди озадачило, что продукт вышел точно в срок и принес прибыль. Последние страницы книги, названные автором как Postmortem, содержат такой текст:

Никогда бы не подумал, что мое знакомство с *Microsoft* закончится летописанием истории провального проекта. Тем не менее с самого начала и до конца моего пребывания здесь меня не покидало ощущение, что мне преподносят урок, как не надо разрабатывать продукт. С тех пор как все, кто был задействован в проекте Explorapedia, казались такими несчастливыми, злыми и беспрерывно говорили, как они расстроены и разочарованы, я мог сделать только один вывод – что невольно созерцаю надвигающуюся катастрофу. Однако по факту проект Explorapedia был безоговорочно успешен.

В следующем предложении Муди оказывается очень близко к тому, чтобы назвать продукт «медведем-плясуном», вот его слова: «Несмотря на то что каждая опция продукта Explorapedia в отдельности являет собой лишь блеклое подобие той опции, что задумывалась первоначально… этой энциклопедии удалось стать единственным продуктом в своем роде на этом рынке». Как же просто оказаться победителем, если конкурентов поблизости не наблюдается, а к вашим услугам мощная поддержка бренда *Microsoft*, крепкие связи с поставщиками и умопомрачительной величины банковский счет.

Безусловно, самым губительным фактором для продукта можно считать его слабую проработанность. Завершая свой рассказ, автор приводит цитату одной из участниц проектирования, Сары Фокс, в тот момент, когда она…

…смотрит на книгу издательства Dorling Kindersley, на основе которой была создана Explorapedia. Сара потрясенно осознает, что, исследуя печатную книгу, читатели чувствуют себя гораздо более свободно, чем при изучении ее компьютерной версии. Хотя изначально предполагалось, что компьютер станет великой силой, избавляющей от всяких ограничений бумажного издания. Книга, по словам Сары, содержала иллюстрации, вокруг которых свободно располагался текст, так что читатели могли исследовать страницы не спеша, имея возможность охватить большие объемы информации с одного взгляда. В Explorapedia же они были вынуждены проходить через множество всплывающих окон, идущих одно за другим, где в каждый момент времени можно было видеть только несколько предложений. Ужасный парадокс ситуации оказался в том, что компьютер ограничивал читателя в разы сильнее книги. «Dorling Kindersley сделало все прямо противоположным образом, а мы превратились в некое подобие привратников, ограничивающих доступ к ресурсам».

В *Microsoft* программисты задумывают, контролируют и разрабатывают все наиболее важные проекты. В своей книге Муди описывает еще один проект – мультимедийный компакт-диск; этот продукт стал в некотором роде исключением, так как «проектировщики» были вовлечены на

каждом этапе процесса разработки. Однако при этом они совершенно не задействовали навыки, которые, на мой взгляд, являются обязательными для каждого проектировщика взаимодействия. Казалось, будто они напрочь пренебрегают всеми важными при проектировании взаимодействия вещами: твердым осознанием, чем в проекте заняты программисты, пониманием принципов и методов проектирования, приемами и инструментами для изучения пользователей. Муди ясно дает понять, что единственное, чем проектировщики оказались полезны для проекта, – это своим острым умом, нескончаемой энергией и чувством прекрасного.

Как следствие, взгляд Муди на эту ситуацию получился искаженным. «Проектировщики должны были набросать как можно больше опций, разработчики – отсечь половину, лишь бы уложиться в срок, а менеджер по продукту – уравновесить обе стороны и вынести вердикт». Любое подобное противостояние обычно заканчивается тяжелым ударом, от которого страдают люди, продукт или компания.

Сотрудники *Microsoft*, которые принимали участие в работе над проектом, остались в таком же неведении, как и Муди. Вот что рассказал Кевин Геммил, старший программист проекта:

Кэролин постоянно твердит, что это «адский проект», а Крэйг беспрестанно повторяет, что он еще никогда не сталкивался ни с чем подобным. А еще Крэйг вечно говорит, что вот тут ошибка и тут ошибка, и вот там мы ошиблись с этой энциклопедией Encarta, и теперь здесь снова все повторяется. И Сара тоже не перестает говорить: «Жизненный цикл этого продукта такой… цикличный». И здесь так с каждым проектом! Мы говорим, что учимся на своих ошибках… и все равно каждый раз снова и снова встреваем в ту же [ненормативная лексика].

Читать эти сокровенные описания от Геммила не менее захватывающе, чем наблюдать крушение поезда. Читатель, не имеющий представления о происходящем в индустрии разработки ПО, может подумать, что все слишком преувеличено, или упрекнуть Муди в выборе неподходящего представителя из этой культуры. Но Геммил здесь – архетип, а потому его поведение весьма характерно. Мне попадались сотни мужчин – и даже несколько женщин, – в точности похожих на него.

Даже при обычных обстоятельствах участники той команды испытывали затруднения при общении с Геммилом. Дизайнеров и программистов *Microsoft* разделяла громадная культурная пропасть. Обычно разработчики просто не могли объяснить дизайнерам даже мельчайшие детали проблемы программирования. И напротив – долгие недели работы дизайнеров над каким-либо аспектом продукта заканчивались тем, что разработчики выносили жесткий вердикт о невозможности реализации указанного.

Несмотря на то что в последние годы наблюдается некоторое улучшение ситуации, два этих противостоящих лагеря все же говорят на разных языках, их взгляды на мир компьютеров с интеллектуальной, культурной, психологической и эстетической точки зрения полярно различаются. Дизайнеры *Microsoft* имели подход, присущий гуманитариям, а разработчики – математикам и деятелям науки. Разработчики считали себя выше дизайнеров, поскольку полагали, что последние обладают неясным бессистемным мышлением и переменчивыми предпочтениями. Дизайнерам, в свою очередь, казалось, что у разработчиков напрочь отсутствует воображение, а кроме того, они консервативны и имеют тенденцию моментально отклонять все предложения по дизайну, даже не желая попытаться найти решение. А ввиду того что таинства программирования были непостижимы для дизайнеров, те никак не могли оценить, насколько доводы разработчиков справедливы и задуманное действительно не подлежит реализации. «Дизайнеры, – как частенько поговаривал Том Кордри, – это обязательно женщины, они любят поболтать, живут в лофтах, предпочитают вегетарианство и носят в ушах дары природы. Разработчики – непременно мужчины, питаются фастфудом, преимущественно молчаливы и говорят вслух только одно: „Неправда“». Еще он мог бы добавить, что методы разрешения конфликтных ситуаций у разработчиков и дизайнеров также различаются. Разработчики порой склонны вести себя как дети, и, когда им в голову взбредает в шутку обстрелять дверь офиса дизайнеров шариками из игрушечного пистолета, их жертвы спешат пожаловаться супервайзеру. Однако будь на их месте такие же разработчики, они открыли бы ответную стрельбу.

Здесь я хотел бы отметить, что *Microsoft* и Муди называют «дизайнерами» тех специалистов, для которых я обычно применяю слово «графический дизайнер». Графическому дизайнеру свойственно развитое чувство прекрасного, образное мышление, умение рисовать различными

инструментами, и такие специалисты тоже принимают участие в каждом без исключения проекте нашей компании, связанном с дизайном и проектированием. Тем не менее графические дизайнеры творят свою магию только после того, как опытные проектировщики взаимодействия завершают значительный пласт работы по концептуальному и поведенческому проектированию.

К слову, раздраженный ответ «Неправда», упоминаемый Кордри при описании манер разработчиков, прекрасно иллюстрирует одно из «качеств высокоэффективных инженеров» По Бронсона: «Это не я неверно ответил, это вы не так спросили».

Муди был весьма хорошо осведомлен о культурных причудах, свойственных исключительно программистам, потому посвятил не один пассаж красочному описанию их несдержанного, заносчивого, придирчивого поведения, только он так и не сумел по-настоящему понять, в чем их ценность. Описывая реакцию разработчика Геммила, который в типологии Кордри должен любить фастфуд, на действия графического дизайнера Кэролин Бьорк, «женщины, предпочитающей вегетарианство», Муди делает в корне неверные выводы:

Со стороны казалось, будто Геммил просто забавляется и подтрунивает над Бьорк, отвечая на ее вопросы, но его поза и поведение, бесспорно, выражали враждебность. Он сидел, напряженно выпрямившись, будто аршин проглотил, раздраженно постукивая по полу одной ногой и барабаня пальцами по столу. Он словно хотел оказаться где угодно, только не здесь. Его отношение к вопросам Бьорк можно было четко отследить по этому стуку об пол ногой и пальцами об стол. Чем о более сложной реализации программной опции шла речь, тем чаще становились проявления раздражения Геммила.

Муди полагает, что Геммил проявлял раздражение из-за «сложности» задачи. Никогда еще он не был так далек от истины. Сложности программисты просто *обожают*. Чем труднее задача, тем больше удовольствия ее решать. Для хорошего программиста сложность является главным мотиватором. Геммил был раздражен, поскольку его не впечатляла перспектива заниматься написанием скучного кода, а кроме того, он не мог контролировать положение вещей, ведь в данном случае окончательные решения оставались за человеком, которого он не уважал, – за Бьорк, специалиста с нетехническим профилем, чьи решения относительно проектирования продукта казались Геммилу нелогичными. Конечно, Геммил никогда этого не признает, он и сам не до конца понимает, что происходит, но он будет прикрываться «сложностью», чтобы его ни в чем не могли упрекнуть.

Тот, кто планирует стать во главе команды разработчиков, должен заслужить их уважение. Работа программистов так чрезвычайно сложна и требует массы усилий, что они будут яростно защищать свою территорию. Каждый, кто предпримет попытку указывать им, что делать, потерпит неудачу, если только не обладает уважением к программистам и знанием их работы вдоль и поперек. В *Microsoft*, как, впрочем, и во многих других компаниях, существует каста программистов и другие «низшие касты». Так вот этим «низшим кастам», по мнению программистов, нечего и надеяться каким-либо образом повлиять на цикл разработки программного продукта.

Однако *Microsoft* является бесспорно успешной компанией, а это, в свою очередь, влечет удручающий побочный эффект. Многие компании пытаются организовать у себя подобие культуры *Microsoft*, в надежде стать столь же успешными. Слепое копирование внешних признаков успешности взамен вызвавших успех обстоятельств – заблуждение довольно распространенное. Это как если бы кто-то увидел, что генерал Джордж Паттон носит при себе револьверы с перламутровыми рукоятками, и ошибочно решил, что одного лишь вычурного оружия достаточно, чтобы стать выдающимся стратегом.

Муди, сам того не ведая, указывает на еще один весьма примечательный аспект нашей культуры разработки. Многие из тех руководителей, что имеют солидный опыт в создании и продвижении программных продуктов на рынке, никогда не обращались к проектированию взаимодействия. Даже при отсутствии проектирования часть их продуктов оказалась успешна, а часть провалилась, при этом процесс создания не изменялся. Так они пришли к выводу, что провал или успешность продукта – дело случая, все равно что купить лотерейный билет. В рассказе Муди все указывало на провал продукта, тем не менее он стал успехом. С компанией *General Magic*, история которой описывалась в главе 6 «Психбольница в руках пациентов», ситуация прямо противоположная – все указывало на успех, а в итоге продукт провалился. Руководители ищут причины провала и неудачи вовсе не там, где следует это

делать, поэтому не видят закономерность и ложно относят результат на волю случая. Вся эта ситуация чем-то похожа на историю с врачами в девятнадцатом веке – тогда еще не знали, что малярию разносит анофелес – малярийный комар, и считали источником болезни другие факторы, в частности, что болезнь переносится по вечернему воздуху и поражает людей случайным образом, а единственный способ противостоять этой смертельной напасти – надеяться на удачу. Позже, когда была установлена верная причинно-следственная связь возникновения заболевания, его удалось быстро победить.

Культурная изоляция

Для большинства компаний по разработке программ характерна такая ситуация, при которой ответственность за разработку наиболее сложных фрагментов программы возлагается на самых опытных программистов. Взамен их обеспечивают неким иммунитетом против досаждающих звонков пользователей, требующих технической поддержки. При поступлении звонка от пользователя его перенаправляют к специалистам службы технической поддержки или младшим программистам. В особых случаях пользователям все же удается пробиться на линию старшего программиста; обычно это происходит, когда им удалось продемонстрировать свою глубокую осведомленность младшему программисту или службе техподдержки. В результате такого отсеивания получается, что чем старше по должности программист, тем реже ему приходится контактировать с обычными, средними пользователями. А точнее, он делает ошибочный вывод, что те пользователи, которым удалось к нему пробиться, и являются типичными.

Приведу пример компании *Sagent Technology*, занимающейся поставками систем управления данными для корпоративных вычислений. Главным экспертом по базам данных в этой компании является Влад Горелик – легендарный программист с невероятно высокими компетенциями. С ним лично доводится пообщаться только тем клиентам, которые способны вести столь же увлеченные беседы о «сегментировании запросов», «декомпозиции задач» и «кубических моделях представления данных», как и он. Оттого вовсе не вызывает удивления, что, по мнению Влада, обычный пользователь Sagent Information Studio глубоко информирован в области баз данных.

Элис Блэр, менеджер компании по продукту Information Studio, в свою очередь проводит значительную часть времени, общаясь с теми, кто может потенциально оказаться покупателем продукта. В ее задачи входит объяснение клиентам возможностей продукта и его базового функционала. В результате у Элис складывается впечатление, что среди пользователей присутствует много таких людей, которые либо совсем не знакомы с продуктом, либо умеют обращаться с компьютером только на базовом уровне. Оттого не вызывает удивления, что Элис считает важной необходимость технической поддержки для большинства клиентов.

Кендал Косби – специалист службы технической поддержки компании *Sagent*. Он не взаимодействует ни с экспертами, ни с неопытными пользователями. Чаще всего он сталкивается с конечными пользователями, уровень которых можно описать как средний. Так как продукт предназначен для упрощения принятия решений, Кендалу приходится постоянно контактировать с аналитиками в сфере финансов и рынка, которые имеют незначительное представление о компьютерах и базах данных, но которым тем не менее требуется иметь доступ к различным хранилищам данных и инструментам аналитики продаж для решения рабочих задач. Ввиду того что указанные специалисты, с которыми общается Кендал, не слишком подкованы во взаимодействии с компьютером, ему представляется удобным, чтобы программа не отображала функционал либо скрывала хотя бы самые сложные опции. Из трех упомянутых специалистов наиболее точно клиента видит Кендал, но у Влада и Элис гораздо больше возможностей повлиять на архитектуру продукта, в связи с занимаемыми должностями.

Существует одна старинная притча, в которой нескольким незрячим людям впервые попадается на дороге слон. Один из них подходит к ноге слона, ощупывает ее и делает вывод относительно объекта в целом, что «это, должно быть, дерево». Другой подходит к слону со своей стороны, ощупывает бок и объявляет, что «это, должно быть, стена». Третий ощупывает хобот слона и говорит, что «это, должно быть, змея». Равно как и эти незрячие, Элис, Кендал и Влад обладают весьма различными представлениями о сущности клиентов, ввиду того что им приходится общаться с непересекающимися подмножествами пользователей. Если не сказать больше – у каждого из них есть собственное эмпирическое подтверждение своих гипотез.

Потому для получения точного объективного портрета потенциального пользователя необходимо заручиться поддержкой человека, который не втянут в ежедневные рутинные процессы как разработки, так и продаж.

Кровный интерес

Один из факторов в культуре разработки ПО, имеющий весомое значение, – уединенность. Программисты обычно работают поодиночке, каждый отдельный фрагмент кода пишет только один программист. Никто не видит, что именно он набирает, а впоследствии этот код, как правило, никто не читает. Читать код, написанный другим специалистом, – это все равно что пытаться разобраться в чужих конспектах, которые представляют собой непостижимые секретные письмена, а вовсе не художественный роман. Процесс программирования является настолько сложным, что требует от разработчика долгой многочасовой работы без перерыва с максимальной целенаправленной концентрацией. Программисты очень чувствительны к своему уединению и всему, что этому сопутствует. Поскольку никто не контролирует, что пишет в коде программист, программисты отдают себе полный отчет в том, что качество кода целиком лежит на их совести. Руководители могут требовать от них должного уровня качества, однако они не станут затрачивать лишнее время и усилия на проверку, что это качество действительно существует. На расшифровку кода может уйти в разы больше времени, нежели изначально было затрачено на то, чтобы его написать. Программисты прекрасно понимают, что от их собственных действий и решений – больше, чем от чего-либо другого, – зависит, что за продукт получится в итоге и насколько будет доволен пользователь. Они несут личную ответственность за успешность конечного продукта. Поэтому их заинтересованность в успехе всего предприятия очень высока.

Такой уединенный характер работы программиста усиливает его ощущение собственной власти. Некоторым программистам такое ощущение вовсе не по душе, однако еще больше они не могут терпеть делегирование полномочий тем, кто не так заинтересован в этом деле. Когда маркетологи, руководители или проектировщики дают им советы, программисты реагируют на них с изрядной долей скептицизма, поскольку знают, что стоит принять совет, который приведет к отрицательному результату, как советчик вмиг испарится, а вся вина падет прямиком на программиста.

Позволять программистам самостоятельно осуществлять проектирование взаимодействия означает в итоге получить не только некачественный проект, но и побочный эффект, при котором программист и вовсе потеряет уважение ко всему процессу проектирования.

Программисты так часто с успехом продирались сквозь процесс проектирования, что уже начали умалять его ценность. В итоге, когда компания все же решает прибегнуть к помощи проектировщика взаимодействия, вполне естественно, что программист относится к его деятельности весьма пренебрежительно.

В конце концов это в целом заканчивается неуважением к проектировщику, процессу проектирования и, как ни печально, к результату этого процесса. Из-за подобного неуважения в культуре компании укрепляется отношение к проектированию как к рекомендации или к смутному совету, нежели как к ясному, детализированному, однозначному указанию. А ввиду того что, на взгляд программиста, его личное мнение не менее ценно, чем какой-то простой совет, он считает допустимым извлекать из проектной спецификации только те детали, которые кажутся ему привлекательными. Он воспринимает задокументированную спецификацию не как строгий план, а как раздел газетной публицистики с письмами и мнениями авторов и читателей. Кое-что из этого весьма занимательно, но далеко от истины, а другое – справедливо, но совершенно ни к чему. К сожалению, принимая подобные решения, программист руководствуется возможностями реализации или личными предпочтениями, а потому его решения часто бывают неверны.

Однако если взглянуть на ситуацию иначе, то программисту знакомы жуткие истории о хороших продуктах, которые вдруг проваливались из-за бестолковых указаний руководителей, в равной степени неосведомленных о потребностях пользователей. Был на моей памяти один руководитель высшего звена, который ненавидел набирать тексты с клавиатуры, а потому требовал, чтобы во всех программах, выпускаемых компанией, было исключительно управление с помощью мыши. Помню я и еще одного руководителя, кто, напротив, имел затруднения в

использовании мыши, а потому заявил, что все программные продукты компании должны быть управляемы только посредством клавиатуры. Этот разрушительный подход к проектированию, в основе которого лежали личные предпочтения, вызвал всплеск полной безысходности в обеих компаниях.

* * *

Бесспорно, существуют и такие программисты, которым свойственно намеренное агрессивное и деструктивное поведение, но среди того огромного количества встреченных мной разработчиков они были столь же немногочисленны, как зубы у курицы. Совершенно закономерно и неизбежно, что, пройдя через испытание длительной интенсивной подготовкой, вынуждающей доходить до пределов человеческих способностей, программисты начинают относиться к другим людям как к менее компетентным специалистам. Разработчики ПО склонны уважать других специалистов, если те не выходят за рамки своей профессиональной сферы, но стоит тому, кто программистом не является, возникнуть в мире программирования – то, как описывает это Муди, – программисты начинают вести себя пренебрежительно и даже высокомерно.

У программиста есть полное право насмехаться над дилетантами, которые посмели сунуть свой нос в высокотехнологичный мир разработки программного обеспечения. Это все равно что программист заявился бы к ревизору и начал пересчитывать бизнес-показатели – ревизор также имеет право высмеять такое самонадеянное и заносчивое поведение программиста.

Сложность ситуации возрастает из-за того, что в типичном процессе разработки проектирование взаимодействия и его реализация переплетаются слишком тесно. И хотя руководитель может потребовать от разработчика внести изменения в поведение программы, он не может заставить его использовать другие методы разработки. Однако именно из-за столь тесной взаимосвязи между поведением программы и его реализацией невозможно затронуть одно, не затрагивая другое. Это одна из тех проблем, которые Муди заметил в период его пребывания в *Microsoft*.

Большинство из тех, кто вовлечен в разработку программных продуктов, желают, чтобы уж их-то творения отличались простотой в применении. В результате они беспрестанно вмешиваются в деятельность программистов, а у тех, в свою очередь, обычно не бывает свободного времени, так что подобные вмешательства их крайне раздражают. Это вынуждает многих из них прибегать к изоляции от остальных и взаимодействовать с прочими участниками команды разработки, не занятых в программировании, лишь в самом крайнем случае. Тамра Хизершоу-Харт однажды рассказала мне, какие приключения ее поджидали, когда она занимала позицию технического писателя и нуждалась в информации от программистов.

Я обнаружила, что подкуп в случае с программистами работает гораздо эффективнее, нежели просьбы. Чаще всего мне в этом помогал шоколад. Метод подкупа оказался настолько хорошим, что однажды руководитель группы разработки на коленях перед всеми умолял меня простить его за то, что он забыл сказать мне о внесении изменений в продукт. (И да, он все равно получил свое лакомство.) А в одной компании мне довелось работать с инженером, который так «подсел» на шоколад, что даже рассказывал мне об изменениях, внесенных его коллегами, лишь бы только самому получить порцию шоколада, предназначенную им. До того как я испробовала метод шоколада, мне приходилось тратить уйму времени, пытаясь выяснить, что же изменилось в продукте. При помощи подкупа мне удалось больше чем в два раза сократить время переработок.

Примечательность этой забавной истории в том, что если мы имеем хоть какое-либо представление о происходящем в сфере разработки ПО, то сразу же признаем, что такое вполне могло происходить. В то время как если бы вам довелось услышать подобную историю про ревизора, которому пришлось подкупать шоколадом агента по работе с дебиторской задолженностью, лишь бы получить от него информацию по сегодняшним депозитам, этот рассказ вас крайне изумил бы, возмутил и вызвал недоверие.

* * *

Для большинства руководящих работников уже привычна ситуация, что их подчиненные незамедлительно реагируют на любое их указание или даже тончайший намек. Им кажется, что раз программисты являются техническим персоналом, то они занимают не такое высокое положение в иерархии, а потому покорно исполнят волю вышестоящих. По мнению же программистов, руководители в данной ситуации не имеют кровного интереса, то есть ничем не рискуют, а потому разработчики не слишком охотно им подчиняются. Разработчик ПО, стремящийся к независимости, не будет вносить изменения в свой код только потому, что кто-то этого потребовал, вне зависимости от ранга данного сотрудника.

Чтобы изменить написанный код, в первую очередь вам нужно повлиять на сознание программиста. Он заинтересован одновременно и в том, чтобы сохранить уже написанное, и в том, чтобы не делать лишней (по его мнению) работы относительно изменения кода. При этом недостаточно потребовать или даже просто попросить, нужно привести рациональный, обоснованный довод в пользу изменений, причем представленный знакомыми инженеру концепциями, и исходить это должно из уст человека, кровно заинтересованного в положительном исходе.

Книга Пола Глена[21] являет собой невыразимо точный анализ, обличающий образ мышления и поведения программистов. Если вы желаете лучше понимать программистов и их культуру, непременно прочитайте эту книгу.

Ограниченное мышление

Один из самых сильных факторов, оказывающих влияние на проектирование программного обеспечения, – это, как я его называю, «ограниченное мышление». В основе этого фактора лежат две взаимодействующие силы. Тот факт, что индустрия программного обеспечения достаточно молода, известен многим, но именно из-за молодости эта индустрия не склонна к самоанализу. Нас так занимает ассимиляция новых технологий, что мы не находим возможности осознать то неправильное представление, сложившееся о более старых технологиях. В результате в индустрии программного обеспечения остается множество мифов и неувязок, которые даже не подвергаются сомнению.

Это удивительно, но тот простой и вполне очевидный факт, что в настоящее время компьютеры стали в разы производительнее, быстрее и при этом дешевле, чем всего лишь несколько лет назад, еще до конца не прижился в практике разработки программного обеспечения. Как следствие, большинство программных продуктов не слишком приспособлены под решение пользовательских задач. Вместо этого они «бросаются» на защиту центрального процессора, ошибочно полагая, что он перегружен. В итоге кто оказывается перегружен, так это пользователь. Вот что говорит об этом подходе Билл Могридж, идеолог проектирования: «Будь милосерден с микрочипами и беспощаден с пользователями».

Невероятный технологический прорыв последних десятилетий превратил высокопроизводительные настольные компьютеры по весьма приемлемым ценам в обычное явление. Теперь каждый студент или домохозяйка могут обзавестись настолько мощным устройством, которому в 1974 году мог позавидовать даже корпоративный центр обработки данных *General Motors*. Несмотря на это мы все еще применяем для создания большинства программ все те же инструменты, технологии, методы и мыслительные парадигмы, свойственные миру «ограниченного мышления». Разработчики все еще по привычке задаются вопросами: «Сможем ли мы соблюсти все требования? Будет ли отклик достаточно быстрым? Чем мы можем пожертвовать, чтобы добиться большей эффективности?» При этом совсем не уделяется внимание вопросам, которые подошли бы здесь куда больше: «Будет ли это понятным пользователю? Можем ли мы представить эту информацию так, чтобы это имело смысл? Достаточно ли ряд этих инструкций отвечает потребностям пользователя? Какая информация важна для пользователя более всего?»

За очень редким исключением, процессоры большинства компьютеров работают вхолостую, то есть пребывают в бездействии. Конечно, ряд вычислительных процессов действительно требует задействовать ресурсы, однако это происходит далеко не так часто, как убеждают нас в этом производители аппаратного обеспечения, стремящиеся продать нам все новейшие, мощнейшие и величайшие технологические чудеса. Совершенно не в их интересах, чтобы

пользователь знал, что его процессор будет бездействовать 75–80 % времени, а подвергаться сильной загрузке лишь на краткие промежутки его функционирования.

Всего каких-то 20–30 лет назад компьютеры обладали такой незначительной производительностью и при этом стоили так дорого, что любая здравая идея ограничивалась малой мощностью головного компьютера. В те времена приоритетным направлением компьютерных наук была разработка технологий, способных уменьшить нагрузку на ограниченные вычислительные ресурсы. Так, для снижения нагрузки применялись такие специально спроектированные технологии, как реляционные базы данных, таблица символов ASCII, файловые системы и язык программирования BASIC. Для программного обеспечения тех лет характерно стремление к высокой производительности в ущерб всем прочим соображениям, в том числе легкости в использовании программы. Однако не стоит забывать, что единожды написанный код обладает поистине природной жизнестойкостью, а потому большая часть этого старого кода, созданного для слабых компьютеров, сегодня лежит в основе программ для современных, высокопроизводительных систем.

Бесчеловечными нас делает процесс, а не технологии

С той поры как на экраны вышел фильм Чарли Чаплина «Новые времена» (Modern Times), широко распространилось суждение, что технологии делают нас бесчеловечными. Я крайне не согласен с такой позицией. Задолго до возникновения технологий тираны, варвары и иные воинствующие племена лишали своих жертв человечности посредством кулаков и камней. Не нужно обладать изощренными инструментами, чтобы ожесточить своего человекоподобного товарища, – для этого достаточно одного взгляда или пинка. Совсем не технологии делают нас бесчеловечными, а технические специалисты, вернее, те процессы, которые они применяют для того, чтобы создавать свои бесчеловечные продукты.

Очевидно, что чем больше потенциал технологии, тем больший вред способен нанести некорректный процесс. Верно и обратное: при правильном подходе к проектированию те же технологии могут превратиться в великий дар человечеству. Сами по себе высокие технологии могут пойти любым путем, но именно человек, который эти технологии направляет, определяет конечный эффект от них.

Интерактивные системы необязательно всегда бесчеловечны, и, чтобы этого не произошло, мы должны перестроить процесс разработки, с тем чтобы в приоритете оставались люди, которые эти технологии используют. Первое, самое важное изменение в этом процессе, которое мы способны совершить, – это выполнять проектирование интерактивных продуктов строго до момента начала их программирования. Второе важное изменение – это привлекать к процессу проектирования специально обученных для этого специалистов, проектировщиков взаимодействия. В ходе нескольких следующих глав я продемонстрирую, каких высот можно достичь, следуя этим шагам.

Часть IV. Проектирование взаимодействия – дело хорошее

9. Проектируем для удовольствия

Альберт Эйнштейн как-то сказал, что невозможно решить проблему, обладая тем же уровнем мышления, который ее породил. Я посвятил уже достаточное количество страниц описанию устаревшего образа мышления и причин, по которым оно не дает эффекта. Теперь же пришла пора обсудить новый подход, который *будет эффективным*. Я разрабатывал его с самого 1992 года и дал ему название «целеориентированное проектирование» (Goal-Directed Design). Проектировщики в моей консалтинговой компании применяют этот метод во всех наших проектах. Он базируется на инновационном подходе к решению проблем, а также содержит некоторые мощные направляющие постулаты и невероятно эффективные приемы мышления. Следующие три главы я посвятил обзору трех самых эффективных инструментов, вместе с несколькими кейсами, демонстрирующими их применение и ожидаемые результаты.

Персоны

Самые мощные инструменты обычно довольно просты по своему устройству, однако воспользоваться ими, как правило, бывает сложно. Безусловно, это характерно и для

инструментов проектирования взаимодействия. Определение нашего самого эффективного инструмента довольно незамысловато: *разработайте детальное описание потенциального пользователя вашего продукта и его намерений*. Трудности начинаются в процессе подготовки деталей такого описания и при его последующем применении.

Здесь сразу представляется очевидным отыскать настоящего пользователя и расспросить обо всем его, однако в данном случае этот подход не работает по нескольким причинам, главной из которых является то, что человек, испытывающий затруднения с какой-либо проблемой, не оказывается по умолчанию наделен видением способа ее решения. При том что настоящий пользователь все еще остается ценным ресурсом и мы посвящаем ему достаточно времени, нам никогда не следует позволять пользователю оказывать прямое влияние на возможное решение.

При всей примитивности указанного метода он является невероятно мощным и эффективным для любой задачи: мы составляем портреты выдуманных пользователей и создаем проект под *их*нужды. Таких придуманных людей мы назвали «персонами» (*personas*[22]) – они составляют фундамент хорошего проектирования взаимодействия.

Персоны не являются реальными людьми, но в ходе проектирования они их олицетворяют. По сути, они представляют собой *гипотетические архетипы* настоящих пользователей. Несмотря на их выдуманную природу, их описание должно быть выполнено с принципиальной строгостью и точностью. В действительности персоны не совсем плод нашего воображения – скорее, это побочный продукт процесса исследования: мы *выявляем* их характеристики. Плодом воображения являются только их имена и персональные данные.

Персоны выделяются в зависимости от их целей. А цели определяются по персонам. Звучит как тавтология, но на деле это не так. Персоны выявляются посредством исследования и анализа приблизительно таким же образом, как и последовательность тектонических событий выявляется геологами после изучения осадочных пород: наличие окаменелостей говорит о том, что здесь присутствует геологический слой, в то же время наличие слоев говорит о присутствии окаменелостей. О целях я буду говорить в достаточном объеме в следующей главе, а сейчас скажу лишь, что они выявляются теми же способами, что и персоны. Подходящие персоны и их цели определяются за счет последовательной детализации в ходе углубления в процесс первичного изучения предметной области.

Обычно мы выявляем характеристики приблизительно, но при этом придерживаясь разумных рамок, а затем быстро определяемся с возможным набором персон. И хотя этот итеративный процесс схож с тем подходом, который применяют разработчики ПО при реализации продуктов, он обладает одним кардинальным отличием. Итерации процесса проектирования и всего, что с ним связано, проходят легко и быстро, потому что мы используем для этого бумагу и слова. Итерации же процесса непосредственной реализации продукта проходят гораздо медленнее и сложнее, поскольку на этом этапе требуется написание кода.

Проектируйте только для одной персоны

При создании продукта, рассчитанного на удовлетворение потребностей широкой аудитории пользователей, логика обычно побуждает наделить его настолько обширными функциональными возможностями, чтобы охватить как можно большее количество людей. *В данном случае логика ошибается*. Ваш продукт станет куда как более успешен, если вы спроектируете его только для одного человека.

Вообразите, что собираетесь спроектировать автомобиль, который понравился бы широкому кругу покупателей. Вы легко выделите как минимум три целевых сегмента: мамочки, вечно возящие своих детей в спортивные секции, плотники и молодые управленцы. Мамочка хочет иметь надежную, безопасную машину, просторную внутри и непременно с большими дверями – чтобы там поместилось все: дети, собаки, пакеты из супермаркета и много чего еще. Плотник Джо хотел бы обзавестись прочным автомобилем с полным приводом и большим пространством для перевозки стремянок, древесины, мешков с цементом и инструментов. Сет, молодой управленец, желает обладать спортивным автомобилем, в котором должен быть мощный двигатель, жесткая подвеска, откидной верх и место только для двоих.

Если исходить из соображений логики, то решение может выглядеть как на рисунке выше. Это некая комбинация пожеланий каждого из трех наших водителей: фургон с откидным верхом, вместительным салоном для детей и местом для перевозки древесины. Какая несуразная, невообразимая машина получилась! Даже если ее действительно можно сконструировать, никто не захотел бы ее купить. Правильным было бы сконструировать минивэн для мамочки, пикап для плотника Джо и спорткар для управленца Сета.

В отношении разработки ПО создать три разных программы гораздо легче, чем сконструировать три автомобиля. Один программный продукт всегда можно реализовать таким образом, чтобы внутри него было три «двигателя» и он обладал поведением трех разных продуктов (с одним только замечанием: нельзя сваливать процесс конфигурирования такой программы на пользователя).

Каждый раз, когда вы расширяете функциональные возможности программы в попытках охватить еще один пользовательский сегмент, вы возводите на пути всех прочих пользователей лишние барьеры из опций и элементов управления. Вы очень скоро выясните, что возможности, призванные осчастливить одних пользователей, препятствуют получению удовольствия других. Пытаясь угодить пользователям со слишком разными потребностями, вы просто-напросто убьете потенциально хороший продукт. И напротив, если вы сузите все многообразие возможностей под потребности одной персоны, ничто больше не сможет помешать ее счастью.

Роберт Лутц, председатель правления компании *Chrysler*, говорил, что порядка 80 % водителей, принимавших участие в фокус-группах, почувствовали крайнюю неприязнь к новому пикапу Dodge Ram. Тем не менее автомобиль был запущен в производство и стал самой продаваемой машиной благодаря тому, что оставшиеся 20 % просто *влюбились* в нее. Вызвать у людей любовь к вашему продукту, пусть они и в меньшинстве, – вот секрет, который приведет вас к успеху.

Чем больше мишень, тем меньше у вас шансов попасть прямо «в яблочко». Если ваша цель – довести уровень удовлетворенности продуктом до 50 %, вы не сможете сделать это, осчастливив каждого из всей широкой аудитории лишь на 50 %. Этого можно добиться, лишь выделив 50 % пользователей из этого круга, и осчастливить каждого из них на все 100 %. Но это далеко не все. Вы добьетесь еще большего успеха, если сосредоточитесь лишь на 10 % вашего рыночного сегмента, но вызовете у них стопроцентный *экстаз*. Такой подход кажется парадоксальным, однако проектирование для *единственного пользователя* – это наиболее эффективный способ сделать широкую аудиторию довольной вашим продуктом.

Чемодан на колесиках и стикеры

Хорошим примером того, насколько эффективным может быть проектирование для одного человека, является чемодан на колесиках. Когда-то этот небольшого размера чемодан с выдвижными колесами и складной ручкой совершил маленькую революцию в целой индустрии багажа, при этом проектировался он вовсе не для всех и каждого. Первоначально он был предназначен лишь для экипажа самолетов – очень узкого сегмента аудитории. Простота устройства этого продукта полностью удовлетворила потребности данной группы потребителей. Однако довольно скоро всю прелесть использования такого чемодана ощутили и остальные путешественники. Его было удобно перевозить через переполненные залы аэропорта, равно как и маневрировать с ним в узких проходах самолета или укладывать в отсек для багажа.

После того как чемодан на колесах заслужил успех у своей целевой аудитории, его вывели и на другие рынки. Сегодня вы можете увидеть в продаже чемоданы на колесах большего размера, дизайнерские, бронированные и детские чемоданы на колесах. В настоящее время найти чемодан без выдвижных колес и складной ручки уже не так-то просто.

А вот вам еще один пример, как инженер по клеевым материалам компании *3M*, Арт Фрай, случайно изобрел одну из самых полезных и популярных на сегодняшний момент офисных принадлежностей, пытаясь решить собственные весьма специфичные проблемы. Когда Арт Фрай пел в церковном хоре, бумажные закладки постоянно выпадали из псалтыри, отчего он каждый раз сбивался. Портить церковную собственность клейкой лентой Арт Фрай не хотел, потому начал искать более подходящее решение. Он вспомнил, что за несколько лет до этого работал над созданием клеевого материала, который в итоге не был пущен в производство из-за недостаточно высокого коэффициента прочности сцепления. Арт покрыл этим неудавшимся материалом поверхность маленьких листков желтой бумаги и сделал из них закладки. Так появились на свет стикеры Post-It Note от компании *3M*.

Довольные пользователи – это невероятно ценный и полезный актив. Фокусируясь на более узком круге потребителей, вы получаете шанс обзавестись по-настоящему преданными фанатами из вашего целевого сегмента. Как упоминалось в главе 5 «Нелояльность клиентов», преданные клиенты – это те, кто станет вашей лучшей поддержкой в трудные времена. Они не просто свернут горы и вброд перейдут все реки, лишь бы заполучить ваш продукт, но они также представляют собой невероятно эффективный маркетинговый инструмент из когда-либо известных, ведь они лично порекомендуют вас своим друзьям. Создав вокруг своего продукта ажиотаж, вы сможете использовать это, чтобы покорить и другие сегменты рынка.

Гуттаперчевый пользователь

И хотя нашей целью является удовлетворение потребностей пользователя, сам термин «пользователь» представляет некоторую проблему. Неопределенность этого термина делает его столь же бесполезным, как бесполезна бензопила для удаления аппендикса. Для проектирования нам требуется более точный инструмент.

Когда я слышал от кого-либо термин «пользователь», это обычно звучало как «гуттаперчевый пользователь» – то есть человек, который вынужден гнуться, растягиваться и подстраиваться под нужды в текущем моменте. Однако это *программы* должны гнуться, растягиваться и подстраиваться под задачи пользователя – вот какой должна быть наша цель. Программисты пишут бесконечное число программ, ориентируясь на этого мифического гуттаперчевого пользователя, хотя его элементарно не существует. Когда программист считает допустимым погрузить пользователя в пучины файловой системы Windows для поиска каких-либо данных, он рассматривает гуттаперчевого пользователя как рассудительного человека, обладающего компьютерной грамотностью и способного адаптироваться к сложившейся ситуации. Или другой случай – когда программист считает допустимым провести пользователя через какую-нибудь сложную операцию при помощи бестолкового мастера, он рассматривает гуттаперчевого пользователя как покорного, наивного, неопытного новичка. Проектирование под таких гуттаперчевых пользователей развязывает разработчику руки, позволяя писать, как они сами считают нужным, лицемерно утверждая, что они делают это «для пользователя». Ваши настоящие пользователи отнюдь не являются гуттаперчевыми.

У программистов есть внушительная система относительно того, как конструировать программное обеспечение. Хороший программист не будет разбрасываться грубыми обобщениями о разных компьютерах и системах. От него нельзя услышать: «На компьютере это будет работать хорошо». О каком именно компьютере идет речь? О какой модели? С какой операционной системой? С какими периферийными устройствами? Также и проектировщик не должен говорить, что «программы спроектированы для пользователя» или «программа будет дружественна пользователю». Когда доводится слышать подобные слова, они кажутся оправданием навязывания собственных интересов разработчика.

В нашем процессе проектирования мы никогда не говорим о таком абстрактном «пользователе», мы имеем в виду совершенно конкретный образ: персону.

Будьте конкретными

Чем конкретнее мы прописываем характеристики персон, тем более эффективны они в процессе проектирования. Так происходит оттого, что при большей детализации они теряют свою «гуттаперчевость». Например, описывая персону Эмили, мы не говорим, что «она пользуется офисным пакетом программ»; мы конкретно обозначаем, что «она использует программу WordPerfect версии 5.1, чтобы писать письма своей бабушке». Мы не можем позволить Эмили просто «ездить на работу». Нет, она должна «ездить на работу на темно-синей Toyota Camry 1991 года выпуска, с установленным пластиковым детским автокреслом серого цвета и уродливой царапиной на заднем бампере». И работа Эмили не просто работа. Эмили «занимает должность специалиста по открытию счетов компании *Global Airways* – Мемфис, штат Теннесси, ее рабочее место находится в отсеке с бежевыми перегородками». Такая детальная характеристика является невероятно мощным инструментом процесса проектирования и коммуникации. Таким образом, каждая из наших персон описывается самым тщательным образом, максимально конкретно.

Как только мы наделяем Эмили уникальными отличительными чертами, происходит удивительная вещь: в представлении проектировщиков и программистов она становится вполне реальным человеком. Мы можем называть ее по имени, и тогда она обретает еще более осязаемую сущность, позволяя разработчикам оценивать предполагаемые результаты проектирования с ее точки зрения. В ходе того, как Эмили становится все менее гуттаперчевой, начинают проявляться ее навыки, ее мотивация и цели, которых она желает достичь. Вооружившись пониманием этого, мы можем изучить ее в свете предметной области нашей программы и определить, на самом ли деле ее можно считать архетипом пользователя. Проведя подобную процедуру несколько раз и получив определенный опыт, проектировщик далее способен правильно формировать образы персон уже с первого раза.

Наделение персоны именем является одним из самых важных шагов для ее успешного выявления. *Персона, не имеющая имени, попросту бесполезна*. В таком случае никто не будет считать ее конкретным реальным человеком.

При прочих равных условиях я стараюсь включать в состав персон людей разных рас, гендерной принадлежности, национальностей и цвета кожи. Тем не менее я предпочитаю использовать образы типичных представителей какого-либо сегмента аудитории, поскольку обратная ситуация может лишь внести ненужную путаницу. Шаблонность персон хороша, если за счет этого образы становятся более достоверными. Соблюдение политкорректности здесь не является моей целью – мне нужно, чтобы все поверили в реальность этих персон. Будь моя персона сиделкой, я с большей вероятностью сделаю ее женского пола, нежели мужского, но вовсе не потому, что мужчины не работают сиделками, а потому, что подавляющее большинство представителей этого рода занятий – женщины. Если мы описываем пользователя как «компьютерного техника», то в виде персоны предстанет «Ник, прыщавый юнец двадцати трех лет, бывший участник школьного клуба любителей аудио и видео», нежели «Хелена, высокая статная красавица, посещавшая частную школу в Беверли-Хиллз». Мне важна правдоподобность образов, а не разнообразие.

Чтобы персона стала еще более реалистичной в представлении участников проекта, я обычно связываю описания с визуальным образом – добавляю каждой персоне изображение. Эти изображения я, как правило, покупаю за небольшую плату на сетевых фотостоках, однако несколько раз мне доводилось использовать для этого быстрые карандашные наброски. Фотографии, если хотите, можно вырезать и из журналов.

Полностью описанная, конкретизированная персона с визуальным воплощением – это очень мощный инструмент. До тех пор пока пользователь не обретет такие точные характеристики, программисты будут воображать, что пользователи похожи на них, или считать их гуттаперчевыми. Явно определенная персона пользователя – это ваш ключ к успешному преодолению склонности разработчиков к искажению характеристик пользователя или пренебрежению ими. Задолго до того, как будет написана самая первая строка кода, качественное описание персоны пользователя станет необычайно эффективным инструментом проектирования взаимодействия.

Персона должна быть гипотетической

Невероятно важно здесь не смешивать точное определение персоны пользователя с реальными людьми. Настоящие пользователи представляют огромный интерес с точки зрения исследований и сбора «сырых» данных, однако для процесса проектирования они зачастую не только бесполезны, но даже и вредны. Это все равно что вино и виноград: хорошее вино, поданное к ужину, будет вполне уместно, в то время как маленькие, жесткие ягоды «каберне совиньон», поданные самостоятельно, только испортят трапезу. Многие ученые, ссылаясь на данные, полученные эмпирическим путем, смешивают настоящих пользователей с гипотетическими персонами, хотя последние гораздо более ценны для процесса проектирования.

Еще одной основной проблемой, связанной с настоящими пользователями, является то, что у каждого реального человека есть свои занятные причуды или поведенческие привычки, отличные от нормы, – все это затрудняет процесс проектирования. Подобные отличительные черты, присущие отдельным личностям, не являются показательными для всей группы. Если один пользователь не приветствует прямое манипулирование объектами на экране, это вовсе не значит, что все остальные – или даже большая часть из них – придерживаются того же мнения. Справедливо и обратное: один из наших настоящих пользователей может вполне успешно справляться с проблемами когнитивного сопротивления, возникающими на его пути взаимодействия с программой, в то время как для всех остальных эта задача невыполнима. Очень велико бывает желание приписать эту способность всем пользователям только лишь потому, что один реальный человек способен на такое, тем не менее этому желанию нужно противостоять.

Чаще всего желание поступить подобным образом возникает у президентов компаний. Так, президент одной из компаний, с которой нам доводилось работать, терпеть не мог набирать текст с клавиатуры, поэтому он требовал, чтобы все взаимодействие было реализовано без ее использования. Он издал приказ о том, чтобы все программное обеспечение в компании управлялось только посредством мыши. Стремиться управлять программами при помощи одной

лишь мыши – вполне разумное желание, однако отнюдь не разумно пренебрегать всеми теми пользователями, которые более комфортно чувствуют себя при работе с клавиатурой. Этого президента нельзя назвать типичной персоной.

Конкретика в описании важнее, чем правильность

Если рассматривать персону как инструмент проектирования, то конкретика в ее описании более важна, нежели правильность. Я имею в виду, что следует определять в подробных и точных деталях отдельные характеристики персоны, – это важнее, чем создание совершенно правдоподобного образа персоны в целом. Такой постулат может показаться удивительным, поскольку он противоречит самой *цели* проектирования взаимодействия, для которой важна достоверность, а не точность. Как бы то ни было, нашей конечной целью является получение программы, способной работать как нужно, так что здесь мы допускаем некоторые разногласия, чтобы только этой цели достичь.

Подвижные звенья механизмов не должны содержать люфтов. Так, при движении поршня в цилиндре зазор должен быть минимален, иначе это приведет к быстрому выведению механизма из строя. Здесь не имеет значения, какова длина поршня, а важна только величина зазора. То же самое справедливо и в отношении персон. Персона должна быть определена достаточно конкретно – так она сможет выстоять под давлением процесса разработки, и в данном случае это важнее, чем обладать правильностью.

Например, в качестве персоны в процессе проектирования чемодана на колесиках мы могли бы описать Герда, командира экипажа «Боинга-747» авиакомпании *Lufthansa*, совершающего рейс из Ванкувера во Франкфурт.

С одной стороны, невозможно расширить эту персону так, чтобы охватить пилотов *всех* коммерческих рейсов. Так, Соня посещает занятия в Университете аэронавтики Эмбри Риддл и по окончании станет профессиональным пилотом. Она совершает ежедневные полеты, но на небольших одномоторных самолетах, и ей никогда не приходится ночевать вдали от дома. Что касается багажа, то Соня здесь представляет собой исключение. Тогда, если расширить описание Герда в попытках включить характеристики Сони, образ персоны станет приблизительным, утратив свою точность. В итоге вы втянетесь в бесконечные и ненужные дискуссии о том, можно ли считать Соню пилотом авиалайнера или нет, и о том, что ей требуется относительно багажа.

С другой стороны, при проектировании чемодана на колесиках мы могли бы с равным успехом выбрать в качестве персоны Франсин, новую стюардессу компании на борту Reno Air. Трижды в день она пролетает через всю Калифорнию, разнося пассажирам напитки и упаковки с арахисом. Личности Герда и Франсин различаются кардинальным образом, однако их цели и требования к багажу одинаковы.

Программисты живут исключениями, а потому применяют тот же подход и при выборе персон. Они будут без конца доказывать, что Соню тоже следует включить в охват персоны, поскольку она сидит в кресле пилота. Но если программирование определяется подобными исключениями и крайними случаями в этой парадигме, то проектирование, напротив, учитывает только центральные события и характеристики. Если есть хоть малейшее сомнение в том, занимает ли персона центральную позицию в парадигме, ее следует вовсе не принимать во внимание.

В целях создания более точных описаний персон следует определить, что считать средней величиной. Средний пользователь фактически никогда на самом деле не является средним в математическом смысле. Средний человек там, где я живу, по статистике имеет 2,3 ребенка, однако фактически такого количества детей не может быть. В данном случае более показательной можно считать персону Сэмюэла – отца двоих детей – или Уэллса, у которого три ребенка. Сэмюэл показателен потому, что является личностью. Конечно, личностью гипотетической, но конкретной. В то время как некий родитель, у которого 2,3 ребенка не может представлять персону с нужной точностью из-за невозможности существования такого среднего количества детей на самом деле.

Персоны с усредненными характеристиками уничтожают все преимущества детализированных персон. Вся мощь персон кроется именно в создании конкретных образов. Всяческие обобщения лишают их этой силы.

Персоны в отдельности наделяют нас мощным инструментом для проектирования взаимодействия. Они составляют основу целеориентированного проектирования. За счет использования персон мы можем осознать масштаб и природу проблем, возникающих в процессе проектирования. Они позволяют с непревзойденной точностью выявить цели пользователя, так что мы можем понять, какие задачи должен решать продукт, а какие решать необязательно. Персона, описанная с нужной точностью, помогает ясно увидеть, каков будет уровень компьютерной грамотности пользователя, так что мы не запутаемся, проектировать ли для экспертов или для новичков.

Персоны, которых мы описываем, должны быть уникальны для каждого проекта. Периодически мы можем обращаться к персонам из прежних проектов, однако с учетом того, что ключевым условием является точность, едва ли можно найти двух совершенно идентичных персон.

Реалистичный взгляд на уровень компьютерной грамотности

Одним из действительно ценных моментов использования персон является то, что они задают более реалистичный тон всем дискуссиям об уровне компьютерной грамотности. Степень подготовленности пользователей может варьироваться в очень широких пределах, и персоны позволяют отчетливо это увидеть. Наиболее распространенная модель уровней компьютерной грамотности в форме пирамиды приводилась в главе 2 «Когнитивное сопротивление». На вершине этой пирамиды находятся «продвинутые пользователи», обладающие абсолютными знаниями о компьютерах, за исключением разве что умения программировать. В середине – «компьютерно грамотные пользователи», понимающие принципы работы компьютеров, но не разбирающиеся во всем многообразии его впечатляющих возможностей. Основание пирамиды – это «неопытные пользователи», которые считаются совершенно недалекими невеждами.

Вот несколько примеров персон, разрушающих ложные представления, взятые за основу при построении пирамиды:

Рупак занимает должность инженера по компьютерным сетям в Лос-Анджелесе. Днями напролет он работает с компьютерами, он эксперт в том, как заставить их функционировать должным образом, тем не менее он не обладает глубинными познаниями, как они устроены. Ему удается держаться на своем месте лишь благодаря практическому опыту, набору суеверий, способности бездумно заучивать новое и безграничному терпению.

Шэннон занимает должность бухгалтера оздоровительного спа-центра в Темпе, штат Аризона. Ей совершенно неизвестно, как работает Всемирная паутина, электронная почта, локальная сеть, файловая система и практически все остальное, что касается компьютера, но при этом она абсолютный гений в использовании электронного табличного процессора Microsoft Excel. В мгновение ока она может сотворить новую таблицу с графиками и диаграммами, отображающую уровень продаж компании.

Декстер занимает должность вице-президента по развитию бизнеса в компании Steinhammer Video Productions, расположенной в Голливуде. Он беспрерывно перемещается между павильонами звукозаписи, потому в карманах его двубортного пиджака умещается множество предметов: пейджер, два мобильных телефона, карманный компьютер и беспроводной модем. Он великий эксперт в технологиях и может решить любую проблему. Его телефон постоянно разрывается от звонков коллег с просьбами помочь им найти пропавшие файлы, однако он действительно слишком занят, чтобы попусту растрачивать время на такие мелочи. Клинт ожидает на третьей линии!

Роберто – телемаркетинговый представитель J. P. Stone, компании-производителя надежной одежды для активного отдыха с доставкой по почте. Его рабочее место находится в офисном отсеке в пригороде Мэдисона, штат Висконсин. Там он надевает свою гарнитуру и принимает телефонные заказы, которые обрабатывает посредством компьютера. Роберто ничего не понимает в компьютерах и высоких технологиях, однако он стрессоустойчивый, трудолюбивый сотрудник, обладающий удивительной способностью с легкостью разбираться в сложных операциях. Всего через несколько дней обучения он стал одним из самых продуктивных и эффективных представителей J. P. Stone. Он говорит, что ему «нравится работать за компьютером».

Самым интересным является то, что ни Рупака, ни Шэннон, ни Декстера, ни Роберто даже близко нельзя отнести к какому-либо из уровней вышеописанной пирамиды. Даже если абстрагироваться от подавляющей власти стереотипов, эта пирамида совершенно не отражает пользовательскую аудиторию. Слишком упрощенные модели рынков никоим образом не помогают решить проблемы проектирования.

С персонами споров о функциях больше нет

Удивительно, но второй чрезвычайно важной и ценной особенностью персон является их способность выступать в качестве великолепного инструмента коммуникации. Набор образов превращается в систему проектирования, обладающую невероятной выразительной силой, позволяющей донести наши представления относительно проектирования. Куда больше они, подобно прожектору, показывают программистам, маркетологам и руководителям, что мы принимаем совершенно верные решения по проектированию.

Жизненно необходимо, чтобы не только каждый участник проектировочной команды ознакомился с набором персон, но и чтобы каждая персона стала почти реальным человеком, словно принимающим участие в процессе разработки. Программисты – с их математическим складом ума – от природы не склонны выделять отдельные случаи, а стараются смотреть на ситуацию в целом, обобщая. Такое восприятие переносится и на их видение в отношении пользователей – в представлении программистов это некие обобщенные, усредненные, универсальные категории. Они видят лишь абстрактного «пользователя», а не «Джуди», «Крэндалла», «Луиса», «Эстеллу», «Раджива» или «Фрэна».

До того момента, как будет разработан набор персон, стандартный диалог между программистом и руководителем, ответственными за проектирование взаимодействия, будет выглядеть приблизительно таким образом:

Программист: «А что если пользователю понадобится распечатать это?»

Руководитель: «Думаю, что в опции печати для первой версии программы большой необходимости нет».

Программист: «Но кому-то может потребоваться возможность распечатки».

Руководитель: «Может, и так. Но мы ведь можем пока отложить добавление этой функции?»

У руководителя практически нет шансов победить в этом споре, поскольку его доводы не выдерживают натиска логики со стороны программиста. Аргументы руководителя, даже будучи истинными, выглядят лишь как невнятное желание сделать все по-своему, в то время как справедливые доводы программиста о том, что «может случиться», непоколебимы.

Разработав набор персон, мы получаем систему, с помощью которой мы совершенно точно можем выразить, кому и какая опция программы нужна. Однако переубедить программистов не так-то легко, поэтому стандартный диалог между программистом, представляющим нашего клиента, и нашим проектировщиком взаимодействия на ранних стадиях рабочих отношений будет выглядеть приблизительно так:

Программист: «А что если пользователю понадобится распечатать это?»

Проектировщик взаимодействия: «Розмари не нужно ничего печатать».

Программист: «Но кому-то может потребоваться возможность распечатки».

Проектировщик взаимодействия: «Мы ведь проектируем для Розмари, а не для этого „кого-то“».

На данном этапе взаимодействия возникает некий конфликт. Программист все еще воспринимает «пользователя» как абстракцию и все еще существует в мире вероятных событий. Тем не менее включение в диалог персоны Розмари позволяет сделать наше намерение менее размытым и неясным. Здесь речь уже идет о конкретном человеке, с определенным набором навыков и целей. Мы наконец-то располагаем убедительным доводом.

Однако из-за того, что доступ к коду имеют программисты, они по-прежнему могут – и будут – делать что им заблагорассудится, вне зависимости от убедительности наших аргументов. И только способность заставить программистов поверить в реальное существование созданных нами персон может стать ключом к успеху. С этого момента каждый из наших проектировщиков начинает решительно настаивать на выражении всех аспектов проектирования в категориях персон с конкретными именами. Мы больше *никогда* не возвращаемся к абстрактному понятию «пользователь». Спустя некоторое время этот трюк срабатывает – программисты начинают

принимать персон как должное и называть их по именам. Казалось бы, перемена незначительна, однако когда программисты делают так по собственной воле, она превращается в поистине сокрушительное событие, кардинальным образом изменяющее всю природу взаимодействия между разработчиками и проектировщиками.

Такой перелом происходит внутри каждого участника наших успешных проектов. Как только это случается, мы резко переключаемся на высокую передачу. Теперь диалог звучит иначе:

Просветленный программист: «Розмари понадобится что-то печатать?»

Счастливый проектировщик взаимодействия: «Нет. Но вот Джейкобу может потребоваться печать отчетов приблизительно раз в квартал».

Просветленный программист: «Ну, если отчеты нужны не так часто, тогда не стоит тратить время и силы на разработку собственного генератора отчетов, а лучше лицензировать стороннее коммерческое решение».

Счастливый руководитель: «Получается, что за счет этого мы сэкономим на разработке целых две недели!»

Я лично наблюдал, какие коренные изменения происходят в компаниях наших клиентов после такого перелома. Ранее они уходили в бесконечные дискуссии о том, какие опции добавить, а уже решенные вопросы снова и снова всплывали на собраниях спустя две недели с момента обсуждения. После же наступления перелома вопросы касательно проектирования возникают и решаются один раз и навсегда, более к ним не возвращаются.

Некоторые из наших клиентов придумали печатать изображения важных персон на футболках для разработчиков. Другие помещали плакаты с персонами над рабочими столами программистов. Подобные меры помогают сплотить программистов в достижении глубокого понимания пользователя их продукта.

Персоны требуются как проектировщикам, так и программистам

Доводилось нам работать и в тех компаниях, где программисты никак не могли привыкнуть называть персон по именам и элементарно не верили в полезность идеи с точными образами. Каждый раз они возвращались к абстрактному «пользователю», отчего самым ужасным образом страдали выпускаемые ими продукты.

Я знаком с одним программистом, который в принципе не понимает, как работают персоны. Он признал важность применения этого инструмента лишь благодаря постоянному давлению со стороны моих коллег и меня. Тем не менее он так и не уловил основную идею касательно конкретики образов, а потому продолжает использовать термин «персона» как синоним слову «пользователь». От него можно услышать такую фразу: «Мы должны удовлетворить потребности персон». И хотя он применяет этот термин, он пренебрегает основной составляющей этого инструмента – конкретикой, вследствие чего этот метод становится совершенно бесполезным.

Был у нас и такой клиент, который отвел нам на подготовку рекомендаций лишь несколько дней. Мы создали образ персоны по имени Эдгар, но не успели наделить его достаточным количеством деталей. После чего мы оказались втянуты в продолжительные обсуждения вопросов, не входящих в исходные рамки проекта. Мы быстро заметили, как Эдгар стал множиться. У каждой команды разработчиков нашего клиента возник свой Эдгар, с собственными отличительными чертами.

Кому удается сразу постигнуть идею применения персон, так это профессиональным маркетологам, поскольку этот инструмент очень схож с тем, чем они занимаются на этапе выявления рыночных сегментов. Главное отличие маркетинговых персон от персон проектирования состоит в том, что образы первых формируются на основе демографических данных и понимания каналов сбыта, в то время как образы последних основаны на пользователях в чистом виде. Это не одни и те же персоны, и цели применения этих инструментов тоже разные. Маркетинговые персоны проливают свет на организацию продаж, а персоны проектирования проливают свет на организацию разработки продукта.

Разрабатывая различные идеи решений для проектирования, мы можем рассматривать их с точки зрения персон и оценивать, насколько каждое решение будет эффективно. Мы сами будто становимся этими персонами, проникая в их разум и глядя на все их глазами. Сделать это легко

благодаря полным и конкретным описаниям образов. Примеряя на себя образ персоны и оценивая продукт или задачу, вы достаточно достоверно можете понять, будет ли такое решение успешным и станет ли ваша персона более счастливой.

Персона отражает пользователя, а не покупателя

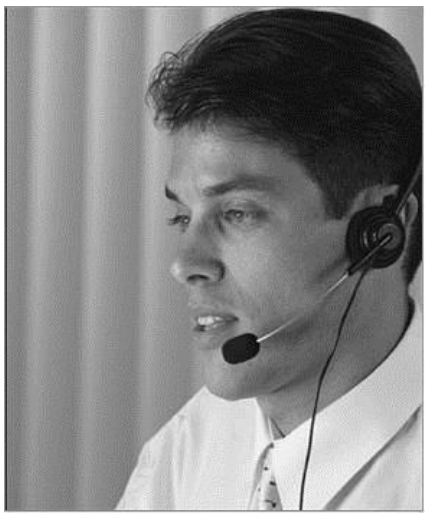
Очень часто проектировщики пытаются создать проект под человека, близкого к продукту, но который на самом деле не является пользователем, – и это одна из самых распространенных ошибок. Так, довольно много продуктов проектируется для журналистов – обозревателей программного обеспечения, пишущих заметки в потребительские издания. То же самое характерно и для сферы информационных технологий – руководитель, занимающийся закупками продукта, едва ли будет тем, кому придется работать с этой программой. Подобная ошибка проектировать для покупателя довольно часто встречается в бизнесе, связанном с разработкой программного обеспечения.

Разумеется, потребностями ИТ-руководителя тоже не стоит пренебрегать, однако он определенно станет более счастлив, если закупаемая программа сможет сделать счастливыми и *конечных пользователей*. Ведь довольство и продуктивность конечного пользователя, как бы то ни было, – это успех и для ИТ-руководителя. Нам частенько доводилось сталкиваться с той ситуацией, когда клиенты игнорируют указанную рекомендацию и сдаются под убеждениями защитников технологий. Когда же действительные конечные пользователи приступают к взаимодействию с продуктом, на руководителей вдруг обрушивается поток жалоб и отказов продолжать работу с программой, на которую ИТ-руководитель возлагал такие большие надежды. После этого руководители обычно переключаются на поставщиков такого программного обеспечения, требуя, чтобы те сделали программу более удовлетворительной для конечных пользователей.

Как создавать образы персон

Для каждого проекта мы составляем отдельный набор образов, включающий от трех до двенадцати уникальных персон. При этом проектирование осуществляется не для всех из них, но каждый образ тем не менее полезен для отражения состава пользователей. Некоторые персоны мы описываем лишь затем, чтобы понять, для кого мы точно проектировать *не будем*. К примеру, один из наших проектов был связан с системой управления технической поддержкой пользователей. Мы определили три персоны – из них два образа отражали внутренних специалистов службы техподдержки. Первой персоной был Лео Пирс – ассистент маркетолога в подразделении компании, отвечающем за продукт. Лео каждый день работает за компьютером и изредка обращается в техподдержку. Вторая персона – Элисон Хардинг, технический специалист компании, обычно переходит из одного кабинета в другой со своим алюминиевым кейсом для инструментов, устраняя неполадки на рабочих местах сотрудников, подобных Лео. Третья персона – Тед Ванверен, специалист службы техподдержки. Каждый день он принимает звонки от пользователей, подобных Лео, и направляет Элисон в тот кабинет, где возникла неполадка.

Перед нашим клиентом, компанией *Remedy Inc.*, в тот момент как раз стояла задача пересмотреть их флагманский продукт Action Request System (ARS) – они хотели добавить системе «легкости в использовании». Разработка образов этих трех персон (а также нескольких других) дала нам возможность четко выразить, какие цели в действительности стояли перед этим проектом.



Теда мы определили как основного пользователя текущей версии ARS, но не он стал нашей главной персоной. Первой мыслью было сделать систему более легкой в использовании для Теда, но, если бы мы поступили так и на этом остановились, это означало бы полный провал. Взамен мы приняли решение обеспечить непосредственным доступом к системе техподдержки персону Лео. Ранее Лео пришлось бы сначала звонить Теду, который затем дал бы указание Элисон. С помощью полного набора образов персон мы смогли ясно представить себе, кто конкретно участвует в этой игре. Так мы сумели донести до разработчиков, что достигнем цели, только если Лео – увлеченный маркетингом и далекий от техники сотрудник – сможет пользоваться системой ARS со своего рабочего компьютера, чтобы напрямую запрашивать помощь технических специалистов, минуя Теда.

Стоило нам описать всю ситуацию в концепциях персон, как участникам команды моментально стало ясно, что нужно сместить фокус внимания с Теда и направить все усилия на Лео. Тед в данном случае становится так называемой антиперсоной (negative persona). Наличие такого образа в наборе помогает определить, под кого проектирование производиться *не будет*.

* * *

Как только у нас появляется образ человека с уникальным набором целей, можно считать, что персона выделена. При этом совсем не обязательно, чтобы все цели всех персон были различны, – вполне достаточно, чтобы совокупность целей каждой конкретной персоны явно отличалась от набора целей других персон. Например, у Рауля – сборщика газонокосилок на конвейере – будут совсем иные цели, нежели у Сесиль, которая осуществляет контроль сборки. Цель Сесиль – увеличить общую производительность и предотвратить возникновение инцидентов. Цель Рауля – выполнить объем работы в разумных пределах, не совершив досадных оплошностей. Хотя в результате они желают получить одно и то же, природа их мотивации различна. Раулю нужна стабильность, а Сесиль – развитие. У них слишком разные цели, поэтому явно видна необходимость выделить две отдельные персоны.

Ключевая персона

Любой набор образов потенциальных пользователей содержит хотя бы одну ключевую персону. Этот главный образ становится центром процесса проектирования. Ключевой является такая персона, которая обязательно должна быть довольна интерфейсом, но при этом она не может быть довольна, если интерфейс спроектирован под какую-то другую персону. Интерфейс всегда делается под ключевую персону. В случае с Remedy ARS такой ключевой персоной стал Лео Пирс.

Этап выделения одной или нескольких ключевых персон при разработке набора образов является жизненно важным. Из своего опыта я могу сказать, что для каждой ключевой персоны требуется свой отдельный и уникальный интерфейс. Если мы выделяем две ключевые персоны, значит, в итоге у нас получится два интерфейса. Выделяем три персоны – будет три интерфейса. Выделяем четыре – значит, что-то идет не так.

Определение более трех персон как ключевых означает, что мы поставили задачу слишком широко и пытаемся за один раз ухватить слишком много. Персоны создаются, чтобы сузить круг возможных конечных пользователей, а значит, если количество персон разрастается неимоверно, получается, что мы забыли об их первоначальном предназначении.

Подбор образов персон – это не фигура речи, это инструмент физической реализации элементов проекта и его логики. Путем отсеивания пользователей, не подходящих для целей проекта, мы в итоге оставляем от трех до семи образов, которые могут оказаться полезными. Затем мы собираем всю информацию о них на одном листе бумаги, включая их имена, фотографии, определение рабочих функций, цели и нередко то, как их описывают другие. Этим одностраничным документом мы руководствуемся в ходе всего процесса проектирования. Распечатки копий этого документа мы раздаем на каждом собрании, вне зависимости от того, присутствует ли на нем представитель клиента. Каждый из наших проектировщиков на каждом собрании, будь то мозговой штурм или разбор деталей проекта, держит перед глазами этот документ со «списком действующих лиц». Когда представители клиента посещают эти собрания, мы печатаем дополнительные копии и для них. Мы вставляем страницу с описанием персон в каждый документ, который создаем для клиента. Всем этим мы упорно добиваемся того, чтобы

персоны стали неотъемлемой частью процесса. Их важность настолько велика, что мы пытаемся внушить эту мысль всем и каждому.

У вас не получится спроектировать продукт качественно, не прибегнув при этом к концепции пользовательских персон. Так вы легко скатитесь до рассуждений об абстрактном «пользователе» и утеряете столь тяжело доставшийся вам фокус на специфичные пользовательские архетипы.

Кейс: система P@ssport от компании Sony Trans Cot

В 1997 году к нам за консультацией обратилась компания *Sony Trans Com* с весьма интересной проблемой из сферы проектирования. Компания *Sony Trans Com* – это подразделение корпорации *Sony*, которое находится в Калифорнии и занимается проектированием и производством систем развлечения пассажиров на борту гражданских авиалиний. Производство подобных развлекательных систем, предназначенных, чтобы пассажиры коммерческих рейсов могли смотреть фильмы в полете, телевизионные шоу, играть в видеоигры, – это огромный и очень высокодоходный бизнес. Компанией *Sony Trans Com* была разработана технология нового поколения, открывающая совершенно новые возможности для пассажиров. Новой системе дали имя P@ssport, а самой впечатляющей из ее опций стала возможность предоставлять видео по запросу (video-on-demand). Эта опция позволяет Трише, занимающей место 23A, начать просмотр фильма «Когда Гарри встретил Салли» спустя 10 минут от начала полета, в то время как Анна, занимающая место 16C, включает тот же фильм спустя 45 минут – и при этом каждая из них может приостановить или перемотать видео, не мешая просмотру другой.

P@ssport вывела системы развлечений в полете на новую высоту, в разы превосходящую все существовавшие на тот момент технические решения. В спинке каждого кресла теперь был встроен видеоэкран и компьютер Pentium под управлением операционной системы Windows 95. Огромный массив медиаконтента хранился на мощном кластере компьютеров, расположенном в носовой части самолета. От компьютеров на пассажирских местах до кластера тянулся оптоволоконный кабель, проходивший через распределительные коробки, установленные по всему салону – по одной на каждые несколько рядов кресел. Благодаря такой организации система работала молниеносно и обладала сверхъестественной мощностью.

До того как обратиться к нам за консультацией по проектированию взаимодействия, *Sony* уже работала над этой системой многие месяцы. Инженеры продвигались в решении своих задач довольно хорошо, а вот проектировщики оказались в тупике. Поскольку в пассажирском кресле мог оказаться практически кто угодно, проектировщики пытались подстроить систему под всех, начиная от абсолютно неопытных пользователей и заканчивая экспертами в компьютерах. Они

не могли понять, каким образом угодить всем клиентам одновременно. Не понимали этого и мы, однако в нашем распоряжении были мощные инструменты проектирования, в том числе персоны, так что уверенность в удачном решении проблемы не покидала нас.

Традиционный подход к проектированию

У компании *Sony Trans Com* уже был готов спроектированный прототип системы P@ssport с типовым интерфейсом, который очень прямо соотносился с внутренней структурой программы, то есть представлял собой модель реализации. По сути, в его основе лежала разветвленная иерархия экранов, по которой осуществлял навигацию пользователь, принимая какое-либо решение на каждом экране. Очевидно, такое устройство прототипа содержало ряд ограничений, что и побудило компанию *Sony* привлечь меня для консультации.

Переход к каждому следующему экрану означал продвижение на еще один уровень иерархии, и для выбора фильма приходилось пройти через шесть экранов.

Здесь мы сталкиваемся с классическим примером того, что я называю «необоснованным выбором», – на каждом шаге пользователю предстоит сделать выбор, масштабы и последствия которого ему не ясны. На первом экране предлагается выбрать вид развлекательного контента: музыка, фильмы, игры, покупки и т. д. Стоит ему нажать Video, как все прочие варианты исчезают, а следующий экран запрашивает желаемую категорию фильма. Так продолжается еще несколько раз, пока в итоге не появляется шестой экран, на котором пользователь может выбрать краткий просмотр отрывка из фильма, чтобы окончательно решить, смотреть фильм целиком или нет. Если на этом шаге он отказывается от просмотра, его откидывает на шесть шагов и на шесть экранов назад к верхнему уровню, где его снова ждут те же шесть шагов при попытке добраться до следующего фильма. Вот это да!

Ввиду того что экраны системы P@ssport встроены в спинки кресел, от пользователей они находятся на расстоянии вытянутой руки. Здесь изначально было ясно, что эти экраны должны быть сенсорными, – это было бы великолепным, само собой разумеющимся решением, в отличие от пульта дистанционного управления, который пользователю пришлось бы держать в руке. Тем не менее эту идею компания *Sony* отвергла. Свое решение они обосновали так: шесть экранов вперед и шесть назад означает, что среднему пользователю придется сделать более десятка касаний экрана на каждый выбор варианта развлечения и что при таком подходе человек, сидящий в кресле впереди, будет невероятно взбешен постоянными нажатиями в области подголовника. Далее произошло то, что По Бронсон описывает как «Они будут чинить и чинить то, что не сломано, пока оно окончательно не сломается». *Sony* пренебрегла идеей сенсорных экранов и вернулась к прежней идее с пультом управления, привязанным к креслу коротким шнуром, чем совершила большую ошибку. Конечно, инженеры об этом решении сожалели, но приняли его как неизбежность ввиду сжатых сроков разработки проекта.

Такой интерфейс с шестью экранами – это типичный пример проектирования по модели реализации, поскольку он в точности соответствовал внутренней структуре программы. На каждом из отображаемых экранов с выбором было так мало поясняющей или дополнительной

информации, что пользователю едва ли удавалось сориентироваться в происходящем, так что навигация была одной из самых больших проблем. Увязая все глубже и глубже в каждом следующем уровне, пользователь терял связь с первоначальным выбором. Стоило нажать Video, как пропадала возможность выбора другого пункта, например Games, – его нельзя было даже увидеть. Каждый шаг не отражал общую картину происходящего, а потому пользователь оставался в неведении. Ему приходилось выбирать Video, не понимая, какие фильмы доступны для просмотра и сколько их вообще. Далее он мог выбрать только одну категорию фильмов, снова не понимая, что, собственно, за ней скрывается. К какому жанру отнести фильм «Правдивая ложь» – к приключениям, романтике или комедиям? Наконец, добравшись до названий фильмов, пользователь и там оказывался лишен всякой надежды на хоть какие-то пояснения. Хм, «Стиратель» – это ведь, кажется, художественная лента о тихом школьном учителе?

Но даже будучи еще прототипом, этот интерфейс уже имел великолепную 3D-графику, красиво отрисованные пиктограммы и концептуальное оформление с картой и земным шаром – все это классическая атрибутика хорошего интерфейса с пустым содержанием. Это пример того, что мы называем «приукрасить мертвеца».

Персоны

Как обычно, мы начали процесс проектирования с проведения обстоятельного исследования, в основном состоящего из ряда интервью с внутренними сотрудниками компании *Sony*. Мы побеседовали практически со всеми, кто принимал участие в разработке продукта, в том числе с руководителем проекта, руководителем разработки, двумя-тремя инженерами, маркетологом по продукту и специалистом, ответственным за медиаконтент системы. В итоге нам стало более чем понятно, каковы ожидания *Sony Trans Com* от данного продукта. Помимо этого, мы получили некоторое представление об историческом контексте развития систем развлечений в полете относительно бизнес-процессов и технологии. Вооруженные этой информацией, мы продолжили исследования уже в полевых условиях – мы выслушали мнения множества сотрудников самолетов, преимущественно бортпроводников с нескольких авиалиний.

В ходе интервью наша коллекция персон пополнялась все новыми и новыми экземплярами. Стоило какой-нибудь стюардессе рассказать нам очередную историю, мы добавляли новую персону, пока их количество не стало порядка тридцати. Чем с большим количеством человек мы беседовали, тем больше информации получали, поэтому вскоре нам явно стали видны сходства отдельных персон. Как только мы обнаруживали персон с общими целями, мы объединяли их в один образ. В конечном счете мы сузили первоначальное количество персон до десяти – получилось четыре образа пассажиров и шесть образов сотрудников авиалиний. Несложно предположить, что наиболее просто было проектировать для сотрудников – их должности можно было описать формально, а в обязанностях легко разобраться. А вот персоны пассажиров оказались действительно «крепким орешком». Каждый образ из четырех составленных нами представлял собой своеобразный архетип, отражая широкий сегмент пользователей, однако проектировать интерфейс для каждого из четырех – невозможно. Требовалось привести их все к общему знаменателю. Вот какой была наша четверка финалистов:

Чак Бургермайстер, находится в деловой поездке. Является членом клуба налетавших 100 000 миль, так как совершает полеты практически каждую неделю. Из-за своего обширного опыта полетов не будет терпеть сложные интерфейсы, отнимающие время, равно как и интерфейсы, предназначенные для совершенных новичков.

Итан Скотт, мальчик девяти лет. Впервые летит один, без сопровождения. Итану нужны игры и только они, в как можно большем количестве.

Мари Дюбуа, летит по бизнес-делам, говорит на двух языках, из которых английский – второй язык для нее. Любит просматривать раздел покупок, а также развлекательные разделы.

Клевис Макклауд, пожилой человек лет семидесяти, упрямый и своенравный, родом из Техаса. Несмотря на преклонный возраст, довольно подвижный, но испытывающий некоторый дискомфорт от небольшого артрита кистей рук. Единственная персона из всех, у кого нет компьютера и кто не умеет им пользоваться.

Пассажиры

Клевис Макклауд
Возраст: 65
Класс обслуживания: World Odyssey

Мари Дюбуа
Возраст: 31
Класс обслуживания: Odyssey Club

Чак Бургермайстер
Возраст: 54
Класс обслуживания: Odyssey Gold

Итан Скотт
Возраст: 9
Класс обслуживания: World Odyssey

Экипаж Odyssey Airlines

Брент Ковингтон
Возраст: 37
Должность: старший бортпроводник

Аманда Кент
Возраст: 28
Должность: бортпроводница

Жан-Поль Дюро
Возраст: 33
Должность: переводчик

Молли Спрингер
Возраст: 41
Должность: специалист по медиаконтенту

Мэл «Хоппи» Хоппер
Возраст: 51
Должность: механик

Джеймс А. Таттерсолл
Возраст: 47
Должность: пилот

Таким образом, наш интерфейс должен был удовлетворить потребности Чака, Итана, Мари и Клевиса, то есть никто из них не должен остаться *недовольным*. Однако это вовсе не значит, что все четверо должны быть безгранично счастливы. Итану известно, что желание играть, играть и снова играть в игры – это не совсем обычная ситуация, так что ему не составит труда нажать несколько дополнительных кнопок, чтобы добиться результата, лишь бы только это в принципе было *возможно*. Чаку известно, что за свой большой опыт полетов он научился делать

некоторые вещи гораздо более быстро и рационально; тем не менее он готов приложить еще немного усилий и изучить специальные команды.

Общим знаменателем в нашем случае оказался Клевис. Компьютера у него не было, как не было и желания когда-либо им обладать. Его девиз: «Старого пса новым трюкам не научишь». Его не назовешь недалеким или ленивым, он просто не входит в число почитателей фокусов новых технологий. Мы понимали, что стоит нам сделать строку заголовка и кнопку закрытия окна видимыми на экране, как мы сразу же потеряем Клевиса. Это означало, что об интерфейсах, похожих на компьютерные, не может быть и речи. Также нам было ясно, что из-за своего артрита Клевис не станет производить сложные манипуляции. Ему должно быть удобно нажимать на кнопки системы основаниями ладони.

Любой вариант, подходящий Чаку, Мари или Итану, был бы неприемлемым для Клевиса. Клевиса испугали бы и запутали клавиши быстрого вызова команд, которые предпочел бы Чак, или возможность выбора языка, что было бы удобным для Мари. А беспрестанно мелькающие перед глазами игры, столь желанные для Итана, попросту ввели бы Клевиса в ступор. Вместе с тем вариант, при котором Клевис – старый упрямый луддит, противник автоматизации – был бы счастлив, совершенно подошел бы Чаку, Итану и Мари, если только их особые запросы также были бы учтены и заложены в какой-либо части интерфейса.

Ввиду того что Чак и Мари долгие годы имеют дело с полетами, они смогут разобраться в любой системе, только бы она не требовала потратить уйму времени на экраны с подсказками для новичков. Нам было понятно, что если мы сделаем систему простой и наглядной, без всяких дополнительных манипуляций, то это не вызовет недовольства у Чака и Мари. С Итаном дело обстояло проще – мы знали, что он достаточно быстро и настойчиво обследует всю систему, чтобы разобраться, где что расположено. Если только его игры не будут глубоко запрятаны, он останется вполне доволен.

В ходе всего процесса проектирования Клевис оставался нашим «пробным камнем». Его фотография стала нашим боевым знаменем. Мы знали, что осчастливить Клевиса – значит осчастливить и всех до единого пассажиров авиарейса. Мы сделали его нашей ключевой персоной и проектировали систему для него и только для него.

Проектирование для персоны Клевиса

Клевис опытом работы с компьютерами не обладал, как не обладал и терпением, чтобы сражаться с программой и получить желаемое лишь спустя некоторое время. Вопрос навигации для Клевиса мы решили достаточно просто: так как он не может и не хочет с ней разбираться, нужно оставить только один экран. А ответом на его нежелание изучать интерфейс стало предложение создать продукт, щедро снабженный пояснениями. Наше решение было скупо на варианты выбора, но богато информацией.

Мы поместили на экране горизонтальную прокручивающуюся галерею, состоявшую из постеров к фильмам и обложек музыкальных альбомов. Мы создали большую круглую ручку, которую назвали «колесо данных», – Клевис мог бы с легкостью крутить ее, аналогично селектору станций на радиоприемнике. Ручка не была изображением на экране, а представляла собой физический элемент управления, размещенный под экраном. Поворачивая ручку в разные стороны, Клевис видел бы, как вместе с этим плавно прокручивается галерея постеров: направо при вращении ручки по часовой стрелке и налево при вращении против часовой.

Так Клевис может просматривать постеры неторопливо, словно прогуливаясь по Бродвею и рассматривая витрины магазинов. Ему не приходится выбирать категорию фильма – и даже думать о том, к какому жанру относится фильм. А ввиду того что разветвленной структуры множественных выборов больше не было, мы вернулись к идее с сенсорным экраном, ведь теперь никому не нужно долбить по экрану, словно дятлу. Если Клевиса заинтересует какой-либо фильм, он просто один раз касается постера и тут же может просмотреть превью, прочесть информацию об актерском составе, ознакомиться с обзорами на эту картину и узнать стоимость просмотра. Затем он может сделать одно касание для начала просмотра или так же в одно касание вернуться к своей неторопливой прогулке по «проспекту кинофильмов».

Постеры к фильмам в прокручивающейся галерее размещены по типу «одноуровневой группировки» – так мы обычно называем единый слой, на котором вся информация разбита по группам. Мы довольно часто прибегаем к такому способу для замены древовидных структур в интерфейсах. Если вы взглянете на свой настоящий рабочий стол, книжные полки или откроете ящики комода – то, вероятно, заметите, что все вещи вы размещаете там по тому же принципу. С такой одноуровневой группировкой очень легко и быстро осваиваются все люди, включая Клевиса, Мари, Итана и Чака. При таком подходе категория фильма перестает быть *обязательным выбором*и становится *полезной дополнительной информацией*. Так у Клевиса появляется возможность присмотреться к постерам до того, как начинать просмотр фильма, и понять, к какой категории этот фильм относится. Это также позволяет решить проблему с отнесением фильма к различным категориям. Например, фильм «Правдивая ложь» может одновременно присутствовать в категории боевиков, приключений, кинодебютов, фильмов со спецэффектами, романтического кино и комедийных лент. Подход, основанный на иерархии, позволил бы отнести фильм только к одной категории, в то время как при одноуровневой группировке все возможные категории могут стать атрибутами фильма.

При прокрутке галереи постеры фильмов плавно сменяются обложками музыкальных альбомов и далее – изображениями к играм. Количество выборов достаточно, но не слишком велико, так что, прокручивая ручку, Клевис может легко добраться до любого элемента. Ручка выполнена крупной, а ее движения точны, так что она будет удобна даже для больших, огрубевших, пораженных артритом рук нефтяника Клевиса. Внизу экрана расположена панель навигации, по которой Клевис может увидеть, какие категории развлечений ему доступны, а небольшой указатель на панели перемещается, обозначая раздел, в котором он находится на данный момент.

Разработчики компании *Sony* не избежали ловушки трех чисел – 0, 1 и бесконечности. Система P@ssport на практике может справиться приблизительно с тремя десятками фильмов. На взгляд программиста, число 30 больше 0 и 1, а значит, оно эквивалентно бесконечности. Реализовать же отображение бесконечного числа фильмов – задача проблематичная, поэтому было принято решение разбить их на группы. Однако Клевису нравится прокручивать эти тридцать вариантов. Будь там даже сто вариантов, все равно он был бы доволен – он бы так же неторопливо «прогуливался», вновь переживая приятные воспоминания о фильмах, которые уже видел, и радостно предвкушал просмотр новых.

Весьма хорошо послужили бы делу и постеры к фильмам – они могли бы отразить важную информацию об актерском составе, сюжете и настроении фильма. Инженерам это было ясно, однако они беспокоились, что необходимость добавлять постеры создаст лишнюю нагрузку на поставщиков медиаконтента. Когда же нам довелось обсудить эту идею с некоторыми из них, они отреагировали прямо противоположным образом. Их невероятно захватила мысль, что постеры к фильмам можно встроить в интерфейс. Они ведь уже потратили сотни тысяч долларов на создание специалистами таких постеров, которые содержат краткие, но максимально информативные описания фильмов, подходящие для самой широкой аудитории. Отчего бы не применить это с пользой для нужд авиакомпании? Они увидели в этом новую прекрасную возможность подготовить растровые изображения постеров для нашего продукта.

Несмотря на то что наш интерфейс был в основном спроектирован под нужды ключевой персоны, мы также не забыли и о потребностях вторичных персон. Так, Чаку Бургермайстеру, частому гостю авиакомпании, было бы удобно иметь под рукой возможность быстрого доступа к опциям, так что мы встроили их в интерфейс, но чтобы Клевису это было незаметно. Когда Чаку потребовалось бы выбрать другой раздел и сделать это быстрее, чем предполагает круглая ручка, он мог бы просто коснуться нужного названия на панели навигации. Тогда программа моментально прокрутила галерею до начала выбранного раздела, без необходимости для Чака делать это самому. Клевис об этой возможности даже бы не задумался, несмотря на ее

постоянную доступность, тем не менее обнаружить и освоить ее очень легко, поэтому пассажиры с более обширным опытом полетов, вроде Чака и Мари, смогут быстро разобраться в системе, изучив ее самостоятельно или наблюдая, как это делают другие.

Физические элементы управления в противовес изображениям на экране явно «призывают» произвести с ними некие манипуляции. Впервые обнаружив круглую ручку, Клевис по форме и расположению сможет догадаться, как ее использовать. Даже с учетом того, что Клевису неизвестно заранее, к чему может привести вращение ручки, стоит ему хоть немного ее повернуть – и он сразу же увидит результат, ведь движение галереи на экране совпадает с движением «колеса данных». Более того, он увидит, как другие пассажиры вращают ручку, а галерея прокручивается вслед за ней. Связь галереи на экране и круглой ручки очевидна, так что Клевис мгновенно освоит работу с системой развлечений.

* * *

До сих пор я подробно описывал лишь интерфейс, который мы спроектировали для персоны Клевиса Макклауда, пассажира авиарейса. Однако мы создали еще два более объемных интерфейса для двух других ключевых персон: стюардессы Аманды Кент и механика Мэла «Хоппи» Хоппера. Их цели отличны от целей Клевиса.

Убедившись в том, что безопасность пассажиров обеспечена, Аманда должна сфокусироваться на предоставлении качественного обслуживания, чтобы у каждого пассажира от полета остались только самые благоприятные впечатления. Потому интерфейс системы для нее должен предоставлять контроль всех операций в полете. Например, если Чак, занимающий место 24C, изъявит желание пересесть, потому что Клевис на месте 24B заснул и громко храпит, у Аманды должна быть возможность переместить счет Чака и наполовину просмотренный фильм на свободное место 19C, куда он пожелал пересесть.

Основным требованием к системе для Хоппи является возможность быстро оценить ее состояние. Он выявляет неисправности, определяет степень их серьезности и принимает решения по их устранению.

И Аманда, и Хоппи используют один и тот же экран, который находится на посту стюардов, однако интерфейсы для каждого из них кардинально различаются, потому как их цели тоже различны.

* * *

Если вы хотите проектировать программные продукты, которые сделают ваших пользователей счастливыми, вам нужно точно понимать, что это за люди. Потому персоны играют такую важную роль в проекте. Следующим шагом будет проектирование настолько мощного продукта, насколько это возможно, а для этого вам потребуется уделить больше внимания целям пользователей.

10. Проектируем для результата

Целеориентированное проектирование, как правило, начинают с формирования набора персон и выяснения их целей. Процесс создания персон я подробно описал в предыдущей главе. В этой я в таких же деталях опишу процесс определения целей и покажу, как эффективно применять их в качестве инструмента проектирования. Два этих аспекта – персоны и цели – неотделимы друг от друга, как две стороны монеты. Суть персоны – в достижении каких-либо целей, а цели, в свою очередь, наполняют смыслом персону.

Мы решаем задачи, чтобы достичь целей

С когнитивным сопротивлением мы столкнулись только с приходом цифровой эпохи, а до ее наступления дизайн и проектирование были понятиями в большей степени из области эстетики, и мнение какого-либо человека в этом вопросе ничем не уступало мнению любого другого. Когнитивное сопротивление возникает там, где есть взаимодействие, а взаимодействие имеет место лишь в том случае, если нужно достичь определенной цели. С таким новым взглядом на ситуацию природа дизайна и проектирования также обрела новый смысл. При этом вопрос эстетики никуда не делся. Он просто очутился в тени более важной потребности – достижения

целей пользователя. Теперь получается, что, в отличие от прежних времен, качество дизайна и проектирования – это не проблема из плоскости субъективных мнений, а тот аспект, который требуется подвергать системному анализу. Другими словами, ясно понимая цели пользователя, мы можем явным образом определить, какой дизайн будет наиболее подходящим для достижения этих целей, вне зависимости от чьего-либо мнения на этот счет, или, если уж на то пошло, эстетических свойств продукта.

Понятие «качественного проектирования взаимодействия» обретает смысл лишь применительно к человеку, который за счет этого пытается достичь своих целей. Не может быть целей без людей. Это неразделимые понятия. Потому центральные элементы нашего процесса дизайна и проектирования – цели и персоны: люди и их намерения.

Куда больше самыми важными для нас являются личные цели конкретного индивидуума. Взаимодействовать с вашим продуктом будет некий реальный человек, а не абстрактная компания, а потому личные цели пользователей следует ставить выше целей компании. Несомненно, ваши пользователи будут прилагать массу усилий для достижения бизнес-целей, но делать это они начнут только после того, как достигнут целей личных. А самой существенной личной целью для них является сохранение собственного достоинства: никто не хочет чувствовать себя глупым.

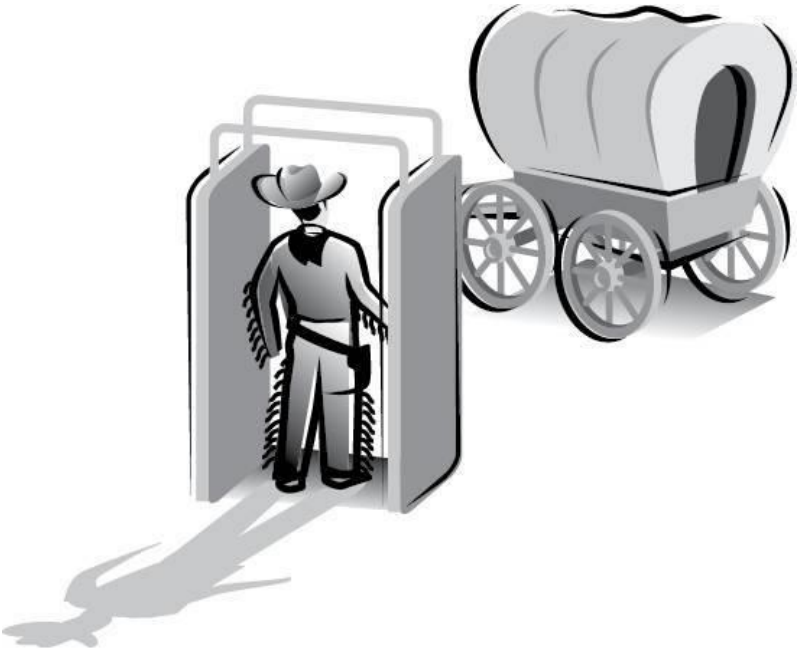
Для качественного проектирования взаимодействия важно разработать такие способы взаимодействия, с которыми пользователи смогут достигать практических бизнес-целей, не пренебрегая целями личными.

Задачи – это не цели

Цели и задачи – это разные вещи. Целью называется конечное состояние, в то время как задачей – промежуточный процесс для достижения этой цели. Очень важно не смешивать два этих понятия, хотя это происходит довольно часто.

Если моей целью является лениво валяться в гамаке, листая воскресную газету, то сначала мне придется подстричь газон. Таким образом, стрижка газона – это задача, а отдых в гамаке – цель. Будь у меня возможность привлечь к выполнению задачи кого-то другого, я достиг бы своей цели, даже не прикасаясь к газонокосилке.

Есть простой способ отличить задачи от целей. Задачи изменяются при изменении технологии, в то время как цели обладают приятным свойством: невероятной стабильностью. К примеру, если я еду в путешествие из Сент-Луиса в Сан-Франциско, то моими целями будут скорость, комфорт и безопасность. Отправляясь в путь до золотых приисков Калифорнии году эдак в 1850-м, я бы сел в свой новенький, высокотехнологичный фургон Конестога[23], а в целях безопасности прихватил бы с собой винтовку. Отправляясь из Сент-Луиса в Кремниевую долину в 1999 году, я бы сел в новенький, высокотехнологичный «Боинг-777», а винтовку в целях безопасности оставил бы дома. Как видите, мои цели остались прежними, а вот задачи с приходом новых технологий претерпели кардинальные изменения, превратившись в прямо противоположные.



Не менее часто встречаются также противоречащие друг другу задачи и цели. Когда президент желает, чтобы за пределами его страны воцарился мир, он направляет туда войска, вооруженные самолетами и боезарядами. Его задачей является развязывание военных действий, а целью – достижение мира. Когда корпоративный юрист желает предотвратить конфликт с коллегой, он начинает оспаривать положения договора. Его целью является достижение согласия, а задачей – спор.

Цель по сути своей стабильна. Задачи же преходящи. Это лишь одна из причин, почему к проектированию под задачи не стоит прибегать во всех случаях, а проектирование под цели уместно всегда.

Подход программистов к проектированию ориентирован на задачи

Достаточно большое количество разработчиков начинает проектировать продукт с попыток найти ответ на вопрос: «Какие задачи он должен решать?» Конечно, это позволяет получить какой-то результат, однако наилучшего решения с таким подходом добиться не удастся, равно как и удовлетворить потребности пользователя. Если проектирование ориентировано на задачи, а не на цели, то оно станет одной из главных причин неэффективного и раздражающего взаимодействия. И напротив, задаваясь вопросом «Какие цели преследует пользователь?», вы сможете выбраться из этой неразберихи и создать более подходящий и удовлетворяющий пользователя дизайн продукта.

Если заглянуть в самую суть компьютерного программирования, то мы увидим, что оно представляет собой создание детализированного пошагового описания различных процедур. Процедура, конечно же, – это инструкция по решению задачи. Хорошие программисты неизбежно обладают процедурным видением проблемы, ориентированным на задачи. В конечном счете задачи должны быть решены, чтобы цели бизнеса были достигнуты, однако задачи могут решаться в разной последовательности и с акцентом на разных их аспектах.

Целеориентированное проектирование

Когда проектировщики производят анализ целей, решая какую-либо проблему, им, как правило, удается найти совершенно другие, гораздо более хорошие решения.

Давайте представим себе Дженнифер, офис-менеджера небольшой компании. Ее целью является обеспечить бесперебойное функционирование офиса. Конечно же, ей также не хочется ощущать себя глупой или совершать ошибки. С такими намерениями ей нужно сделать так, чтобы компьютерная сеть в офисе работала без нареканий. Для этого Дженнифер, во-первых, должна правильно настроить сеть, во-вторых, мониторить ее функционирование, а в-третьих,

периодически изменять ее конфигурацию таким образом, чтобы добиться максимальной производительности. Получается, что со стороны Дженнифер работа представляет собой интеграцию трех указанных задач, выполняемых для достижения единой цели – обеспечения бесперебойной работы офиса. При этом в представлении Дженнифер три этих задачи неотделимы друг от друга. Она не отличает первоначальную настройку сети от дальнейшего изменения ее конфигурации.

А теперь представим себе Клэнси – разработчика программного обеспечения, который должен написать программу для Дженнифер. Со стороны Клэнси и присущего ему образа мышления *Хомо логикус*, ПО для Дженнифер выполняет три задачи – три основные функции, каждая из которых выполняется в отдельном программном модуле. Для Клэнси кажется вполне естественным, что у каждой из этих функций также будет свой раздел в интерфейсе. Такой подход кажется ему логичным. Клэнси задумывает интерфейс, в котором левая боковая панель будет содержать иерархический список компонентов системы, а правая часть экрана будет отображать детали о каждом компоненте, как только он будет выбран из списка. Именно такая форма интерфейса принята в *Microsoft*, а потому программистам она кажется вполне разумной. Конечно, в таком случае пользователь будет вынужден пройтись по множеству компонентов системы, чтобы выяснить, где кроется проблема, однако вся нужная информация уже присутствует в программе и доступна *по запросу*.

И вот настало время для Уэйна – проектировщика взаимодействия, который способен враз осчастливить и Дженнифер, и Клэнси. Уэйн обладает сознанием дизайнера и проектировщика, а потому отчетливо понимает, что программное обеспечение должно предстать перед Дженнифер в таком виде, который наиболее точно соответствует ее целям, но при этом гарантирует, что вся необходимая функциональность будет на месте. (В данном случае Дженнифер является ключевой персоной.) Также Уэйн отдает себе отчет в том, что не может включать в спецификацию технические решения относительно интерфейса, невыполнимые или неразумные с точки зрения программиста Клэнси.

Уэйну совершенно ясно, что цель у Дженнифер только одна: бесперебойная работа офиса, так что он проектирует такой интерфейс, в котором Дженнифер может с одного взгляда точно определить, что все работает стабильно. Если же где-то возникает «узкое место», в интерфейсе появляется яркое, наглядное уведомление о конкретной возникшей проблеме, что дает возможность Дженнифер исследовать и устранить неисправность путем непосредственного взаимодействия с отображением проблемной области на экране. Уэйн также знает, что для Дженнифер не существует разницы между мониторингом системы и изменением ее конфигурации, а потому интерфейс отражает и это представление тоже. С отдельными компонентами системы Дженнифер может понадобиться взаимодействовать в одном-единственном случае – когда ей точно известно, что на то есть серьезные причины.

В представлении Клэнси код, отображающий поведение компонентов сети и код для настройки этих компонентов, – это две отдельные процедуры. Мышление, ориентированное на задачи, не видит между ними взаимосвязи, тогда как для целеориентированного мышления они неотделимы друг от друга. Дженнифер не станет изменять конфигурацию компонента сети, пока для этого не возникнет серьезного повода, такого как сбой в его работе. В дальнейшем же Дженнифер потребуется иметь возможность постоянно наблюдать за поведением этого компонента с измененными настройками.

Проектирование под цели персоны пользователя позволяет нам явно увидеть альтернативные способы предоставления функциональных возможностей. Как правило, с помощью такого подхода получается найти в разы более хорошие способы решения типичных проблем проектирования. Вот несколько примеров.

Целеориентированный выпуск новостей

Клиент, с которым мы взаимодействовали по одному из наших проектов, занимался разработкой пакета приложений для создания новостных телевизионных передач. Со стороны инженера, который обычно мыслит в плоскости задач, создание такой телевизионной передачи похоже на постройку моста: она выстраивается фрагмент за фрагментом. Однако позже нам удалось установить, что такой способ создания телепередачи не был целью ведущих. Напротив,

они хотели, чтобы у них всегда уже была под рукой готовая телепередача, которую они бы имели возможность дополнять с течением времени. Каждая новостная телепередача похожа на живое существо, невероятно изменчивое, сразу рождающееся взрослым и пребывающее в таком состоянии до самого конца.

В бизнесе новостей может случиться всякое, так что ведущим нужна позиция для отступления. Их целью является всегда иметь приемлемую передачу, пригодную для вывода в эфир. Выпуск вечерних новостей создается утром и представляет собой полноценную, готовую к передаче в эфир запись. Ее продолжительность составляет 22 минуты (без учета рекламы), и она всегда готова к показу. Под каждый новостной фрагмент отведено определенное время, а все фрагменты в сумме всегда составляют 22 минуты. Это похоже на то, как изображение с нечетким фокусом медленно обретает резкость: границы телепередачи всегда фиксированы, а контент постепенно уточняется и обрастает подробностями. Передача готова к выпуску уже с десяти утра, ее можно поставить в эфир в любой момент, если потребуется, однако своей наилучшей формы она достигнет приблизительно к пяти часам вечера.

Каждый выпуск новостей включает от 20 до 30 сюжетов, составленных из комментариев, видео, репортажей и интервью в студии. В ходе текущего дня приоритеты сюжетов сдвигаются, равно как сдвигается время выхода каждого и отведенное под него время, – все это делается на усмотрение директора службы новостей. К началу второй половины дня все внимание может привлечь к себе сенсационная новость, из-за чего другие новости придется сдвинуть или вовсе исключить из выпуска. Репортеры и директор службы новостей будут вносить коррективы и изменения в сценарий вплоть до самой последней секунды – иногда даже прямо во время эфира.

Результатом работы разработчиков, воспринимающих эту проблему со стороны задач и процедур, стала программа, с помощью которой новостные выпуски можно было создавать сюжет за сюжетом. Это было невероятно логично и рассудительно, но столь же невероятно неправильно. Выпуск новостей оставался незаконченным вплоть до момента выхода в эфир, а внесение изменений в какой-либо фрагмент разрушительно влияло на другие фрагменты, вследствие чего передача оставалась непригодной для показа, пока все фрагменты вновь не были собраны.

В ходе нашей работы по проектированию мы подготовили наброски программы, взаимодействие с которой начиналось с готовой к выходу в эфир передачи. Репортеры и директор службы новостей могли непрерывно вносить свои коррективы, ровно так, как они делали до этого в ручном режиме. Но в отличие от ручного метода, наша программа позволяла задействовать также всю мощь компьютера. Так, например, если какой-то новостной сюжет исключался из передачи в последний момент, время, отведенное на него, автоматически распределялось на оставшиеся сюжеты в порядке приоритета.

Целеориентированное управление учебным процессом

В другом нашем проекте мы работали над проектированием системы управления учебным процессом для учителей начальной школы. Разработчики создали компоненты для организации тестирования учеников, отслеживания успеваемости и доступа к базе поурочных разработок. Исходя из понимания задач, такое решение представлялось подходящим. Мы же затем, если так можно выразиться, проникли в голову учителей, чтобы выяснить потребности типичного учителя начальной школы. Ответ, который мы там нашли, был поистине поразительным.

Мы выяснили, что учителя, сидя в своих учебных классах, чувствуют себя в некоторой изоляции и страстно желают получить обратную связь на свои действия. Также в целях повышения своего педагогического мастерства учителям требуется получить оценку собственной «успеваемости». И эта потребность совсем не очевидна, если разбивать процесс преподавания на задачи, из которых он состоит. Однако эта человеческая потребность становится отчетливо видна, стоит нам задуматься о целях. В нашем решении по проектированию мы предложили возможность отслеживать достижения учителей по семестрам и классам. С таким инструментом учителя могли лучше видеть свой прогресс и общий ход процесса, что положительно повлияло на уверенность в том, что они делают.

Практические и личные цели

Ранее в этой главе я говорил о том, что суть качественного проектирования взаимодействия заключается в том, чтобы дать пользователям возможность достигнуть практических целей, не пренебрегая целями личными. Представители вида *Хомо логикус* и апологеты высоких технологий считают несколько нескромным слишком сильно задумываться о личных целях пользователя, а потому всячески этого избегают. Как бы то ни было, различение двух этих видов целей существенно для достижения успеха.

Для примера расскажу про своего коллегу Теда. Недавно он прислал мне электронное письмо, в котором жаловался на свой новый телевизор. Он целый час потратил на чтение инструкции, чтобы лучше разобраться с великим множеством его настроек. Он высказал мне идею, что на экране телевизора должен высвечиваться диалог, который помог бы ему пройти через все шаги настройки вместо нудного чтения инструкции. Предложенное им решение справедливо, однако, не будучи проектировщиком взаимодействия, он выдал вариант в духе прежней эпохи механических устройств: фокусируясь на задачах. Такой диалог на экране телевизора упростил бы задачу установки нужных параметров, однако если бы мы вместо этого использовали иной подход и сфокусировались на целях Теда, мы смогли бы найти в разы более качественное решение.

Оценивая цели Теда, нужно идти от самого очевидного. Нам несомненно известно, что Тед хочет посмотреть телевизор. А поскольку он отдал за этот ящик уйму денег, то не менее очевидно, что он хочет иметь возможность воспользоваться всеми его невероятными встроенными функциями. Это практические цели, и они самым явным образом связаны с задачей настройки телевизора.

Однако здесь мы совсем не должны забывать о том, что Тед – живой человек с собственными предпочтениями, которые также можно считать целями. Тед совсем не обрадуется, если новое приобретение заставит его чувствовать себя унизительно или глупо. И еще Тед не хочет совершить ошибку. Для него важно ощутить результат, и чем скорее это произойдет, тем лучше. И он хочет, чтобы все происходило весело. Эти жизненные цели Теда невероятно важны. Со стороны проектировщика взаимодействия эти цели даже важнее, чем практические.

Стоит заметить, что жалобы Теда относились вовсе не к тому, что он не может посмотреть новый телевизор, или что он заплатил за него слишком большую сумму, или что у него не получается воспользоваться всеми его возможностями. *Он жаловался, что чувствует себя глупым, пытаясь разобраться в настройках телевизора*. Разумеется, он сказал это в несколько иной форме, потому что, даже признаваясь кому-то в том, что «эта вещь делает из меня дурака», человек чувствует себя некомфортно, но общий смысл письма Теда был именно такой. Разбираясь в настройках, Тед непреднамеренно совершал ошибки. После того как он включил телевизор в розетку, ему понадобилось более часа, чтобы добиться от устройства хоть какого-то результата. Да и весь процесс настройки параметров телевизора едва ли можно назвать веселым.

Несмотря на то что продукт обладал взаимодействием, отражающим практические цели Теда, он совершенно пренебрегал его личными целями. Та специфика нового продукта, превратившая его в классический пример высокотехнологичной новинки, похожей на пляшущего медведя, характеризует не то, как продукт отвечает практическим целям пользователя, а то, как он *терпит поражение* в удовлетворении личных потребностей человека.

Теперь, отдавая себе отчет в том, что личные цели Теда – это святое, мы можем спроектировать совершенно другой интерфейс для этого телевизора. Для начала, желая быстро наделить владельца телевизора чувством завершенности и достижения результата, мы должны гарантировать, что телевизор начнет работать сразу, как только будет включен. При этом нет необходимости в том, чтобы работали *сразу все* его невероятные возможности, – достаточно, чтобы работала только *часть из них*, но работала хорошо. Становится ясно, что проведение Теда через весь процесс настройки телевизора не отвечает требованиям этого простого теста на получение быстрого удовольствия. Для разработчиков программного обеспечения все функции устройства одинаково бесценны, а потому они не делают различий между ними в интерфейсе. Тем не менее не так сложно выделить часть настроек для первичного использования устройства, а все прочие оставить для последующего знакомства пользователя с устройством, что даст ему некоторую отсрочку по времени, при этом устройство будет функционировать. Все настройки нужно разбить на категории. Решение этой проблемы – вопрос не технической реализации, а расстановки приоритетов при проектировании.

При описанном подходе к проектированию телевизор уже станет похож на потенциально успешный продукт: Тед вынет устройство из коробки, подключит к сети электропитания и сразу же сможет расслабиться в своем кресле, удовлетворенно переключая каналы. Так он достигнет большей части своих практических целей, не пренебрегая целями личными.

Важно отметить, что *отсутствие препон* на пути достижения личных целей гораздо существеннее попытки охватить сразу все практические цели. Различие между двумя этими подходами явно показывает, что «проектирование для кого-то» и «обеспечение функционалом» – это две разные вещи. Решение относительно дизайна и проектирования взаимодействия должно обеспечить Теда всеми необходимыми функциональными возможностями для достижения практических целей, при этом конкретно дизайн интерфейса должен четко показать способы, которые помогут Теду достичь его личных целей.

Принцип соизмеримости усилий

Разумеется, спустя некоторое время желание Теда достичь всех своих практических целей с помощью невероятных возможностей нового телевизора вновь начнет расти и крепнуть. К этому моменту он посвятит телевизору уже много счастливых часов, освоится с ним и будет готов прилагать дополнительные усилия. Теперь телевизору будет не так-то просто унизить Теда, ведь тот стал гораздо терпимее к взаимодействию с устройством и лучше понимает, чего еще он желает добиться от телевизора.

Научно доказано, что люди склонны реагировать на компьютеры эмоционально (об этом чуть больше будет рассказано дальше по тексту). В ходе взаимодействия с устройствами люди наделяют их человеческими чертами. Так, Тед захочет приложить еще немного усилий для настройки телевизора, поскольку чувствует, что телевизор тоже вложил свои усилия в то, чтобы сделать *его* – Теда – более счастливым.

Этот феномен я называю принципом соизмеримости усилий. Люди готовы вкладывать свои силы в решение задач, так как чувствуют, что это справедливый обмен равных с равными. Другими словами, пользователи соглашаются потратить дополнительное время, поскольку ожидают в ответ получить дополнительное вознаграждение.

Личные цели

Давайте остановимся на целях более подробно. Я уже упоминал два вида целей: личные и практические, однако помимо них существуют еще корпоративные и ложные цели. Личные цели просты, схожи у разных людей и, как и следует из их названия, отражают личность человека. Как это ни парадоксально, но именно из-за этих свойств целей людям сложно о них говорить, особенно если это соотносится с рабочими делами.

ЛИЧНЫЕ ЦЕЛИ

Не чувствовать себя глупым

Не допускать ошибок

Выполнить оптимальный объем работы

Провести время весело (или, по крайне мере, не скучать)

Апологетов здесь в особенности беспокоит пункт «не чувствовать себя глупым». Таких людей отличает крайняя уверенность в себе, высокий интеллект и стремление преуспеть в решении сложных задач. Что ж, это звучит вполне в духе высокотехнологичных предпринимателей из Кремниевой долины. Так, когда я любезно отправил Теду – успешному, независимому, высокотехнологичному предпринимателю – историю о телевизоре, записанную с его собственных слов, ответ был таким:

Я бы не сказал, что сражение с 40-страничной инструкцией к телевизору заставляет меня чувствовать себя глупым. Здесь, скорее, речь идет о нежелании усложнять всю ситуацию и тратить лишнее время на решение ненужных задач, а точнее – на изучение тех вещей, в которых впоследствии очень вероятно придется разбираться снова. (Например, если в доме отключат электричество, не придется ли снова задавать настройки для телевизора по той же самой инструкции?)

Тед – явный представитель апологетов. Малейшее бранное слово в адрес техники из его уст поставит под сомнение его способность совладать с телевизором, невзирая на сложность этой задачи. Он готов признавать свое раздражение, пустую трату времени и бесполезные дополнительные действия, но только не тот факт, что техника заставляет его чувствовать себя

глупым, даже если это только намек на глупость. Потому я так настаиваю на этом слове. Я употребляю слово «глупый» именно по той причине, что компетентному, умному, серьезному высококлассному эксперту в программном обеспечении из Кремниевой долины так *сложно* его произнести. Как можно услышать от них самих: первый шаг к решению проблемы – это признать, что она существует.

Личные цели всегда являются истинными и так или иначе применимы для каждого человека. Личные цели всегда преобладают над любыми другими целями, хотя из-за их слишком близкого соотнесения с личностью человека о них так редко говорят. В тот момент, когда программа вынуждает пользователя чувствовать себя глупым, его самооценка резко падает, а производительность работы начинает стремительно снижаться, вне зависимости от всех прочих целей. Любая система, пренебрегающая личными целями пользователя, в конечном счете потерпит неудачу, невзирая на то, как хорошо она позволяет достигать иных целей.

Корпоративные цели

Каждая корпорация выдвигает собственные требования к программному обеспечению, и уровень этих требований столь же высок, как и уровень личных потребностей отдельных пользователей. Весьма распространенной среди советов директоров и акционеров компаний является цель «увеличить прибыль». Чтобы сохранять фокус на глобальных вопросах и не распыляться на более мелкие задачи или ложные цели, проектировщик руководствуется следующими целями.

КОРПОРАТИВНЫЕ ЦЕЛИ

Увеличение прибыли

Увеличение доли рынка

Обход конкурентов

Рекрутинг большего количества сотрудников

Расширение продуктовой линейки или количества услуг

Выход на IPO

Психологи, занимающиеся изучением организации рабочих мест, используют термин «гигиенические факторы». Сол Геллерман[24] описывает эти факторы как «необходимые для эффективной мотивации, однако не являющиеся мотивационными сами по себе». К примеру, освещение в офисе – это гигиенический фактор. Вы не ходите на работу только потому, что там прекрасное освещение, но, если бы света не было вовсе, вы бы и не посмотрели в сторону этого офиса.

Я немного видоизменил этот термин и назвал его «гигиеническими целями». Я определяю их как цели, «необходимые для эффективного функционирования, но не являющиеся мотивационными сами по себе». Все цели, приведенные в списках корпоративных и практических целей, гигиенические. Для корпорации эти цели, безусловно, важны, но не корпорация двигает бизнес, а люди, поэтому их личные цели куда важнее.

Между корпоративными и личными целями можно провести параллель: оба вида целей выражают высочайшие намерения тех, кто их ставит. Значимость обоих видов целей ни в коем случае нельзя принижать. Программное обеспечение, пренебрегающее каким-либо из двух этих видов целей, обречено на провал.

Практические цели

Практические цели являются тем связующим звеном, которое объединяет устремления компании и личные цели каждого пользователя. Если корпорации необходимо, чтобы все сотрудники выкладывались на максимум, чтобы увеличить итоговую прибыль, то практическая цель удовлетворения потребностей клиентов связывает корпоративную цель увеличения прибыли с личными целями пользователя работать более продуктивно.

ПРАКТИЧЕСКИЕ ЦЕЛИ

Избегать совещаний

Удовлетворить потребности клиента

Хранить данные о заказах клиента

Описать бизнес в цифрах

Нередко практические цели представляются более привлекательными, чем личные, в особенности это касается отношения к делу у предпринимателей и программистов. В

соответствии с природой своего внутреннего устройства эти люди создают программное обеспечение, которое не способно удовлетворить потребности отдельных пользователей, хотя в достижении практических целей помогает идеально. Интерфейс с ориентацией на задачи вынуждает пользователя совершать ошибки, что снижает их продуктивность, заставляет думать о себе плохо и с неприязнью относиться к программам.

Естественно, что для достижения целей бизнеса программное обеспечение должно обладать рядом встроенных в него определенных функций. У пользователя должна быть возможность решать задачи, связанные с обработкой клиентских запросов и заказов, однако эти цели являются гигиеническими, так как добавление соответствующих функциональных возможностей без учета личных целей пользователя не принесет успеха. Не достигая личных целей, пользователь не сможет достичь и целей компании. Общеизвестно, что счастливые и всем довольные сотрудники работают наиболее эффективно. В условиях современной информационной экономики эта истина становится еще более справедливой, ведь настоящими активами компании являются люди, а не механизмы. Есть и обратная сторона: если ваша программа служит *исключительно* достижению личных целей пользователя и пренебрегает практическими целями, то вы только что спроектировали компьютерную игру.

Ложные цели

Большинство из тех программных продуктов и устройств на их основе, что мы используем каждый день, сконструировано на основе ложных целей. Эти цели – по крайней мере, многие из них – делают процесс разработки программ легче, тем самым помогая программистам достигнуть их собственных целей, отчего таким ложным целям отдается приоритет в ущерб конечным пользователям. Другие ложные цели касаются задач, возможностей и инструментов. Все это способы достичь результата, но сами по себе они результатами не являются, в то время как цель – это всегда результат.

ЛОЖНЫЕ ЦЕЛИ

Сэкономить память
Меньше использовать клавиатурный ввод
Запускать в браузере
Упростить изучение
Обеспечить целостность данных
Увеличить скорость обработки данных
Повысить производительность программы
Задействовать новейшие технологии или возможности
Улучшить внешний вид компонентов
Обеспечить поддержку совместимости с другими платформами

«Обеспечить целостность данных» вовсе не является целью для программы организации персонального списка рассылок в той же мере, что и для программы, рассчитывающей орбиты движения космических шаттлов. Цель «сэкономить память» не является такой уж критичной для работы с базами данных на персональных компьютерах, поскольку объемы данных не столь велики, а компьютеры обладают достаточной мощностью. И даже цель «упростить изучение» не всегда считается основной. Например, если пилот истребителя с легкостью изучит боевые системы самолета, а затем внезапно обнаружит, что они совершенно неудобны из-за медленной скорости и неуклюжести, то в реальных условиях воздушного боя он окажется в незавидном положении. Целью пилота является победа в бою, а не легкий отдых во время учебного инструктажа.

С тех пор как был изобретен микропроцессор, компьютерная революция стремительно несется вперед на гребне волны новых технологий. Компания, которая относится к высокотехнологическим идеям с пренебрежением, обречена на забвение. Однако здесь не следует смешивать способы и цели. Использовать новые технологии – это *задача* компании, но это никогда не станет *целью* пользователя. Как пользователю мне совершенно безразлично, решу ли я свои рабочие задачи с помощью баз данных, какими бы они ни были – иерархическими, реляционными или объектно-ориентированными, – с помощью плоской файловой системы или посредством магических заклинаний. Мне важно лишь, что работа будет выполнена – быстро, легко и с достоинством.

Так, компания *Visioneer Company* в 1996 году перетянула на себя существенную долю рынка настольных сканеров, обойдя довольно сильных конкурентов. Этот невероятный подвиг компании удалось совершить при помощи старомодных черно-белых сканеров, в то время как у конкурентов на рынке были сканеры, обладавшие возможностью передачи изображения в полутонах или даже в цвете. У компании *Visioneer* был продукт, в комплект поставки которого входило программное обеспечение, спроектированное на основе целеориентированного подхода, позволявшее пользователям выполнять сортировку полученных изображений и с легкостью просматривать их, тогда как продукты конкурентов только лишь сохраняли изображения в запутанной файловой системе.

Компьютер тоже человек

Профессора Стэнфордского университета Клиффорд Насс и Байрон Ривз занимаются изучением реакций людей при взаимодействии с компьютером. Применяя к этой новой сфере классические эксперименты социальной психологии, они смогли отследить в какой-то мере удивительные особенности поведения. По результатам своих исследований Насс и Ривз издали книгу под названием *The Media Equation*[25] («Формула цифровых данных»). В ней авторы приводят убедительные доказательства того, что реакция людей на компьютеры схожа с их реакцией на других людей.

Так, они пишут, что «эволюция человеческих существ отстает от развития технологий двадцать первого века» и что «современные подходы к представлению цифровых данных задействуют устаревшие приемы… Следствием этого является тот факт, что люди склонны относиться к каждой информационной системе на их пути как к человеку, хотя и понимают, что это несколько странно, и впоследствии, вероятно, станут такое поведение отрицать». В представлении человеческого разума компьютеры больше похожи на людей, чем на деревья и камни, а потому мы неосознанно склонны воспринимать их как живых мыслящих существ, хотя и понимаем, что такое суждение противоречит здравому рассудку.

Другими словами, люди на каком-то особом инстинктивном уровне определяют, как вести себя среди других разумных существ, а потому стоит только *незнакомому* объекту проявить наличие существенного когнитивного сопротивления, как наши инстинкты заставляют мгновенно реагировать на него как на живое существо, способное мыслить. Проявление подобной реакции происходит бессознательно и неизбежно, это характерно для всех людей. Особая глубокая и неподражаемая ирония всей ситуации заключается в том, что Насс и Ривз привлекли к участию в эксперименте с компьютерами студентов – выпускников технических специальностей, довольно опытных в обращении с различными устройствами и самостоятельно создававших тестовые программы. Все эти высокообразованные, зрелые, рационально мыслящие индивидуумы как один упрямо отрицали тот факт, что они попали под эмоциональное влияние когнитивного сопротивления, хотя доказательства этому были неопровержимы.

Нейробиолог и когнитивист Стивен Пинкер дает подтверждение этому умозаключению в своей примечательной книге *How the Mind Works*[26] («Как устроен наш разум»). Он пишет: «Людям присуща вера в то, что противоречит их практическому опыту, однако является справедливым для той среды, которая окружает их на каждом этапе развития. Они преследуют цели, которые ухудшают их собственное благополучие, но зато в полной мере соответствуют среде их обитания».

Проектирование и обходительность

В этом исследовании есть один важный вывод, которому стоит уделить особое внимание: если нашей целью является создать желанную для пользователя программу, мы должны проектировать ее поведение так, чтобы программа вела себя как симпатичный нам человек. Если мы хотим добиться от пользователей эффективной работы с программным продуктом, мы должны спроектировать его так, чтобы его поведение походило на поведение хорошего сотрудника. Это совсем не сложно, не так ли?

Насс и Ривз говорят о том, что программа должна обладать «обходительностью, вежливостью», поскольку именно такое поведение широко распространено в социуме (свод правил касательно вежливости отличается в каждой культуре, однако он несомненно присутствует в любой из них). Таким образом, создавая продукты с высоким коэффициентом когнитивного сопротивления, мы должны последовать этому несложному правилу и наделить их

вежливым и обходительным поведением. Для многих программ характерно считать слова «спасибо» и «пожалуйста» компенсацией любой потенциальной грубости в адрес пользователя, однако это совершенно не тот подход, который можно назвать обходительным.

Если программа утаивает часть информации, вызывая у пользователя затруднения в оценке происходящего, заставляет его тратить время на выискивание самых важных и часто используемых функций и в придачу норовит обвинить пользователя в собственных промахах, пользователь отнесется к такой программе с неприязнью и получит негативный опыт взаимодействия. В этом случае даже присутствие в интерфейсе слов «спасибо» и «пожалуйста» будет не способно исправить ситуацию. Не скроет недостатки и идеально красивый, визуально насыщенный, чрезмерно информативный и даже антропоморфный интерфейс.

И напротив: если программа ведет себя крайне обходительно, если все взаимодействие с пользователем происходит в уважительном и вежливом тоне, а интерфейс содержит необходимую информацию, пользователю будет приятно работать с такой программой и он получит позитивный опыт взаимодействия. Здесь стоит еще раз подчеркнуть, что тип интерфейса на результат взаимодействия не влияет: если командная строка на зеленом фоне будет обладать указанными характеристиками, пользователи примут такой вариант.

Что такое обходительность?

Что же представляет собой дружественная пользователю, или обходительная, программа? Как сделать ее поведение более схожим с человеческим? Так, люди, продающие подержанные автомобили, могут выглядеть привлекательно, одаривать нас широкими улыбками и красиво говорить, но станет ли от этого взаимодействие с ними более приятным? Также людям свойственно ошибаться, медленно думать, совершать импульсивные поступки, однако из этого не стоит делать вывод, что программа с перечисленными характеристиками получится хорошей. Есть особые качества, которые делают некоторых людей наиболее подходящими для работы в сфере обслуживания. Программы всегда занимаются именно обслуживанием[27].

Многих хороших разработчиков программного обеспечения необходимость проектирования обходительности застает врасплох. Роберт Кринджели описывает программистов так:

Им присуща выразительность и невероятная точность, но только если они сами заинтересованы в этом. Они выглядят так, как выглядят, вовсе не из-за лени, а потому что таким образом они подчеркивают свои личные приоритеты. Манера их общения крайне сухая, из-за чего кажется, будто они совсем не склонны к коммуникациям. Стоит вам назвать его Майком, в то время как он сам называет себя Майклом, и вы не дождетесь от него ответа, так как с его точки зрения вы обращаетесь не к нему[28].

Вы можете своими глазами увидеть, как «обходительность» или даже «человечность» становится камнем преткновения, если попросите программистов интерпретировать эти понятия

со столь неясным смыслом. Они начинают отчаянно сопротивляться идее сделать компьютеры более человечными, поскольку, с их точки зрения, люди – это маломощные и несовершенные вычислительные устройства.

Я как-то задал вопрос своему другу Киту Плису – известному в профессиональном сообществе разработчиков успешному программисту и эксперту, очень чувствительному ко всем тем аспектам, которые касаются пользовательского взаимодействия. Вопрос касался того, как сделать программы более человечными. Кит истолковал концепцию «добавления программам человечности» как «уход от конкретики при взаимодействии». Его ответ был таким:

По-твоему, компьютер должен лгать пользователю? Сообщать, что на его банковском счету осталось «приблизительно 500 долларов»? Или отвечать иначе, чем только что ответил кому-то другому? Если мы добавим человечности, наши системы станут меньше похожи на компьютеры, по крайней мере, в сравнении с текущей ситуацией.

Относительно программиста, каким является Кит, такая реакция вполне естественна. Действительно, компьютеры не склонны сообщать нам ориентировочные значения сумм на банковских счетах. Однако они также не видят разницы между тем, чтобы за десятую долю секунды выдать ответ «приблизительно $500» или ответ «ровно $503,47» за 17 минут. По-настоящему обходительная, человечная программа моментально сообщит пользователю, что на его счету «приблизительно $500», а затем проинформирует его, что отобразит более точную цифру через несколько минут. В этом случае решение, потратить ли эти лишние несколько минут на ожидание, останется за *пользователем*. Так применяется принцип соизмеримости усилий: если вам требуется больше информации, вы согласитесь затратить дополнительное время для ее получения.

Что делает программы обходительными?

Есть много признаков, по которым людей можно назвать «обходительными», однако определить эти признаки конкретными словами не так просто. По словам Насса и Ривза, «существует четыре принципа, которые составляют основу правил обходительного взаимодействия: качество, количество, значимость и прозрачность». Указанные принципы хороши, однако это не слишком ясные определения, чтобы из них можно было извлечь какую-то пользу. Я приведу собственный список того, что улучшает качество взаимодействия; он подходит и для людей, и для высокотехнологичных программных продуктов, обладающих высокой степенью когнитивного сопротивления.

Обходительная программа интересуется мной.
Обходительная программа относится ко мне с почтением.
Обходительная программа ведет себя приветливо.
Обходительная программа обладает здравым смыслом.
Обходительная программа предугадывает мои потребности.
Обходительная программа обладает отзывчивостью.
Обходительная программа не спешит жаловаться на личные проблемы.
Обходительная программа всегда в курсе происходящего.
Обходительная программа обладает проницательностью.
Обходительная программа характеризуется уверенностью в своих силах.
Обходительная программа концентрируется на важном.
Обходительная программа позволяет обойти правила.
Обходительная программа поощряет незамедлительно.
Обходительная программа вызывает доверие.

Обходительная программа интересуется мной

Друзья обычно интересуются нами – тем, кто мы такие и что нам нравится. Они помнят о наших предпочтениях и о том, что мы не любим, чтобы в будущем можно было сделать нам приятное. Любая компания, которая оказывает услуги с последующим сопровождением, приложит определенные усилия, чтобы научиться запоминать лица и имена своих клиентов. Кому-то нравится, когда его называют по имени, кому-то не очень, однако у всех без исключения вызывает восторг отношение, которое учитывает личные предпочтения человека.

Большинство же программ не ведают, кто ими пользуется, и не прилагают усилий, чтобы исправить эту ситуацию. Ни одна из *персональных* программ на моем *персональном* компьютере фактически не помнит ни меня, ни какой-либо информации обо мне. И это действительно так, невзирая на то, что этой программой пользуюсь исключительно я и делаю это постоянно и беспрерывно. Ларри Кили как-то пошутил, что писсуару с автоматическим смывом в уборной аэропорта есть больше дела до него, чем его персональному компьютеру.

Каждый бит в любой из программ, установленных на моем персональном компьютере, должен усердно работать над запоминанием моих рабочих привычек и в особенности над тем, какие настройки я в них устанавливаю. Программист, который пишет подобные программы, считает, что вся информация в компьютерном мире должна поступать точно в срок, а потому, если программе требуются хоть малейшие данные, она просто-напросто полагает, что их можно запросить у пользователя. Через мгновение беспечная программа отбрасывает эту информацию, рассчитывая на то, что при необходимости запросит ее снова. Дело не только в том, что компьютер, в общем-то, и предназначен для хранения информации, но также в том, что забывать что-либо просто невежливо.

Так, в адресной книге моего почтового клиента записано одиннадцать человек по имени Дэйв. Почти со всеми из них я общаюсь крайне редко, за исключением одного – моего лучшего друга Дэйва Карлика, я постоянно пишу ему письма по электронной почте. Создавая новое сообщение, я ввожу в поле адресата имя Dave, для которого может быть много соответствий. Но я ожидаю, что программа проанализирует мои прошлые действия и на основе этого анализа сделает вывод, что я подразумеваю Дэйва Карлика. Если бы я хотел написать письмо какому-либо другому из известных мне Дэйвов, к примеру, Дэвиду Фору, я бы ввел Dave F, D4, David Fore или как-то иначе, в целях обозначить, что сейчас мой выбор отличается от обычного. Однако программа пренебрегает моими привычками и каждый раз выдает диалоговое окно, в котором мне приходится уточнять, кому из одиннадцати Дэйвов я хочу отправить письмо. Я безразличен этой программе, она относится ко мне как к незнакомцу, в то время как на самом деле я – единственный, кого она знает.

Обходительная программа относится ко мне с почтением

Любой сотрудник из сферы обслуживания относится к своим клиентам с почтением. Ему известно, что клиент всегда прав и его пожелания должны быть учтены. Если метрдотель в ресторане провожает меня к столику, я рассматриваю его выбор лишь как предложение, а не обязанность. Выражая протест в вежливой форме и склоняясь к выбору другого столика в пустом зале, я ожидаю, что к моему желанию немедленно прислушаются. Если в ответ последует отказ, я, скорее всего, покину этот ресторан и выберу другой, где моим желаниям отдадут больший приоритет, чем желаниям метрдотеля.

Необходительная программа осуществляет постоянный контроль над действиями человека, воспринимая его как недостаточно компетентного. При этом допустимо, когда программа высказывает *предположение*, что я совершил ошибку, и недопустимо, когда она оценивает мои действия. Аналогичным образом допустимо, чтобы программа *подсказывала*, что я не смогу отправить введенные данные до тех пор, пока не укажу номер социального страхования. Тем не менее, если я все же решу нажать кнопку «Подтвердить отправку» введенных данных, не указывая этот номер, я ожидаю, что программа выполнит то, что положено. (Сама надпись «Подтвердить отправку» (кнопка Submit) и то понятие, которое стоит за этой фразой, уже выражает неуважение к пользователю. Программа не должна сомневаться в действиях пользователя, поэтому каждый раз, когда она просит подтверждения, она де-факто проявляет неуважительное отношение. Тем не менее кнопки с такой надписью очень часто встречаются на сайтах Всемирной паутины.)

Обходительная программа ведет себя приветливо

Когда я спрашиваю работника аэропорта, какой выход нужен пассажирам рейса 79, я жду, что он не только ответит на мой вопрос, но и самостоятельно сообщит мне крайне полезные дополнительные сведения о том, что указанный рейс задерживается на двадцать минут.

Когда я делаю заказ в ресторане, то автоматически подразумеваю, что мне потребуются нож, вилка, ложка, стакан воды, соль, горчица и салфетка.

Программы в большинстве своем не обладают ни одним из подобных умений. Они способны давать сухие ответы на конкретные вопросы и скупы на дополнительную информацию, даже если она непосредственно касается моих целей. Когда я делаю попытку распечатать документ, то установленный на моем компьютере текстовый процессор ни за что не уведомит меня о том, что в принтере закончилась бумага или что мой документ сорок первый в очереди, хотя обходительный человек сразу же сообщил бы мне об этом.

Обходительная программа обладает здравым смыслом

В каждом хорошем ресторане гостю любезно устроят экскурсию по кухне, однако здравый смысл подскажет любому хозяину, что гостя следует вместо этого сопроводить в зал, чуть только он появится у входа. При использовании программных продуктов нередко складывается впечатление, будто программы не различают «кухню» и «зал», ставя самые часто используемые опции в один ряд с теми, которые задействуются редко. В меню такой программы, помимо простых и безвредных функций, можно легко наткнуться на те, что представляют «смертельную опасность», действия которых необратимы, вроде тех, что катапультируют пилота из кабины. С такими функциями способны работать только профессионалы со специальной подготовкой. Это все равно что усадить гостя за обеденный стол прямо возле пышущего жаром гриля.

Размещение неподходящего функционала в неподходящих местах – вот особенность программных продуктов и устройств на их основе. Система бесключевого доступа моего автомобиля – великолепный пример подобного отсутствия здравого смысла. А вот ранее упомянутый пример с ответом, что на моем счету «приблизительно $500», – это хорошая иллюстрация обратной ситуации, где здравый смысл присутствует.

Известно немало жутких историй о пользователях, оскорбленных иррационально рациональными компьютерными системами, беспрестанно присылающими им то чеки на сумму $0,00, то счета на $8 943 702 624,23. После благоразумной изоляции конечных потребителей от компьютерных систем кошмары служб поддержки клиентов закончились, однако большинству внутренних сотрудников компаний все же приходится взаимодействовать с подобными системами. Сотрудники получают плату за свои рабочие часы, а потому пытаются не выражать свои жалобы слишком громко, да и пожаловаться некому – служба поддержки клиентов обычно не принимает заявки от собственных сотрудников компании.

Обходительная программа предугадывает мои потребности

Моя ассистентка в курсе, что, если я улетаю в другой город на конференцию, мне требуется гостиничный номер. Она понимает это, хотя я не даю ей явного указания на этот счет. Также она знает, что я отдаю предпочтение тихим номерам, предназначенным для некурящих, а потому именно такие и бронирует для меня, без дополнительных распоряжений с моей стороны. Она предугадывает мои потребности.

Мой браузер бездействует большую часть времени, пока я просматриваю сайты в интернете, хотя он тоже мог бы предугадать мои потребности и подготовиться к ним, а не тратить время впустую. Почему бы ему не использовать это время простоя с пользой и не начать загрузку страниц по ссылкам в зоне его видимости? Ведь с большой долей вероятности я обращусь к этим страницам спустя какое-то время. Невостребованный запрос я смогу прервать достаточно быстро, в то время как на ожидание исполнения нового запроса может потребоваться дополнительное время. Будь у программы способность предугадывать мои желания и готовиться к ним, а не дожидаться, пока я отдам ей такое распоряжение, она выглядела бы куда более обходительной и при невысокой скорости интернета.

Обходительная программа обладает отзывчивостью

Когда я обедаю в ресторане, то ожидаю от официанта адекватной реакции на мои невербальные намеки. Когда же я веду оживленную дискуссию с моими соседями по столику, я ожидаю, что официант найдет себе другое занятие на это время. Было бы крайне неуместным, если бы официант вдруг ворвался в нашу беседу и сказал нечто вроде: «Добрый день. Меня зовут Рауль, я ваш официант на сегодняшний вечер». Если же наша дискуссия завершилась и я поворачиваю голову, пытаясь поймать взгляд Рауля, то ожидаю, что он поспешит к нашему столику, чтобы услышать мои пожелания.

Стандартный видеорежим, в котором работает экран моего компьютера, – 1024x768 точек. Когда я провожу презентации и подключаю компьютер к проектору, мне приходится временно изменять разрешение экрана, устанавливая более низкое значение 800×600 точек. Во многих из тех программ, что я использую, включая и саму операционную систему Windows 2000, при изменении разрешения уменьшаются размеры, внешний вид и положение окон на экране. Впоследствии я неизменно и быстро снова устанавливаю разрешение экрана в 1024x768 точек, однако окна, которые до этого подстроились под более низкое разрешение, не спешат вновь принять размеры, подходящие под высокое разрешение. Компьютер обладает для этого всей необходимой информацией, тем не менее ему совершенно нет дела до моих вполне очевидных потребностей.

Обходительная программа не спешит жаловаться на личные проблемы

От барменов, врачей и парикмахеров в барах, кабинетах психотерапевтов и салонах красоты мы ожидаем, что они будут помалкивать о собственных проблемах, но проявят здоровый интерес к нашим. Возможно, такая однобокость представляется не слишком честной, но именно такой является природа сферы услуг. Так же и программы должны молчать о себе и показывать явный интерес ко мне. Поскольку компьютеры лишены самолюбия и не испытывают нежных чувств, они могли бы сыграть роль идеального наперсника, однако их поведение обычно свидетельствует о прямо противоположном отношении.

Программы вечно жалуются мне посредством своих диалогов подтверждения и бахвалятся своими ненужными и неудобными строками состояния. Для меня нет нужды выслушивать, как тяжко компьютеру выполнять свою работу, эта информация является для меня лишней. Меня не волнует, что программа испытывает проблемы с уверенностью в себе, не решаясь очистить мусорную корзину. Я не желаю слушать нытье по поводу того, что ей неизвестно, куда именно сохранить файл на диске. Для меня нет необходимости слушать писк модема и наблюдать информацию о ходе передачи данных или процессе загрузки точно так же, как я не хочу выслушивать о разводе бармена, сломанной машине парикмахера или о том, что врач должен платить алименты.

Здесь стоит уточнить два момента. Программе не только следует молчать о своих проблемах, но она также должна быть в достаточной мере сообразительной, уверенной и компетентной, чтобы самостоятельно справиться с этими проблемами.

Обходительная программа всегда в курсе происходящего

И напротив, каждый хочет обладать информацией о происходящем. Тот же самый бармен помогает мне быть в курсе событий, помещая прейскурант на видное место или записывая на меловой доске время субботнего празднования в честь начала матча вместе с информацией об игроках и текущем счете.

Владельцам магазинов следовало бы уведомлять покупателей о том, что может быть для них особенно значимым. Мне не нужно, чтобы мясник, к которому я хожу постоянно, сообщал мне о том, что все индейки для Дня благодарения закончились ровно в сам День благодарения. Эта информация нужна мне заранее, чтобы я понимал, что запас ограничен, и спланировал покупку заблаговременно.

Когда я ищу информацию в интернете при помощи обычной поисковой системы, у меня никогда нет уверенности в том, в какой момент ссылка окажется для меня бесполезной из-за ее неактуальности. Я кликаю по ссылке на тему, которая меня интересует, и все, что получаю в ответ, – это отвратительное сообщение «Ошибка 404» (страница не найдена). Почему бы

поисковым системам не осуществлять периодическую проверку ссылок на актуальность? Если страницы по ссылке больше не существует, поисковая система могла бы исключить эту ссылку из списка, так что я не тратил бы на нее лишнее время.

Программы беспрестанно предлагают мне какие-либо опции, недоступные в настоящий момент по неизвестным причинам. Программа должна самостоятельно контролировать такие опции и не предлагать их мне.

Обходительная программа обладает проницательностью

Консьержка в одном из отелей Нью-Йорка, где я останавливаюсь довольно часто, запомнила, что я интересуюсь бродвейскими постановками. Теперь, стоит мне вновь посетить этот отель, консьержка, без каких-либо просьб с моей стороны, кладет на столик в моем номере буклет с текущим репертуаром Бродвея. Она в достаточной степени восприимчива к моим желаниям, благодаря чему может предугадывать их и обеспечивать меня всей нужной информацией еще до того, как я успею об этом подумать. Такая проницательность требует от консьержки совсем небольших усилий, но в результате я снова и снова останавливаюсь именно в этом отеле.

При работе с какими-либо приложениями я всегда разворачиваю их во всю ширину экрана, а затем переключаюсь между окнами при помощи панели задач Windows. Тем не менее мои приложения, видимо, этого совсем не замечают, особенно те из них, что я установил недавно. Мне постоянно приходится разворачивать их самому, хотя уже давно стоило бы обратить внимание на мои явные и однозначные предпочтения. Другие пользователи, напротив, отдают предпочтение меньшему размеру окон, чтобы можно было видеть значки на «Рабочем столе». Для программы не составило бы большого труда замечать подобные предпочтения пользователя и в дальнейшем подстраиваться под них.

Обходительная программа характеризуется уверенностью в своих силах

От сотрудников сферы обслуживания, с которыми мне доводится взаимодействовать, я ожидаю смелых и уверенных действий. Заметь они, что я вышел из уборной, забыв застегнуть ширинку, я был бы признателен, если бы кто-то из них мгновенно отреагировал и ясно, но незаметно для окружающих дал мне это понять, пока я не вошел в конференц-зал и не начал произносить речь. Для таких действий требуется смелость, но такая смелость будет оценена по достоинству. Аналогичным образом, если моей личной ассистентке не удается забронировать для меня билет на подходящий рейс, я ожидаю, что она уверенно подыщет что-то на замену и мне не придется вникать в эту проблему.

Когда я говорю компьютеру удалить какой-либо файл, мне не нужно, чтобы он спрашивал, уверен ли я в своем решении. Естественно, я уверен, иначе не отдавал бы ему такого приказа. Мне нужно, чтобы компьютер действовал смело, в соответствии со своими убеждениями, и просто уничтожил этот файл.

С другой стороны, если у компьютера есть хоть малейшие подозрения, что я совершаю ошибку (а такие подозрения у него есть постоянно), то он должен быть всецело готов к тому, что я могу передумать, после чего потребуется произвести восстановление удаленного файла. Какая бы ситуация ни случилась, программа должна действовать уверенно, а не изворачиваться и не ныть, пытаясь переложить всю ответственность на меня.

Часто бывает так, что я подолгу работаю с документом, затем отправляю его на печать кнопкой Print и ухожу, чтобы налить себе кофе, пока документ выводится на принтер. Когда я возвращаюсь, то обнаруживаю посредине экрана бессмысленное диалоговое окно от насмерть испуганной программы: «Вы уверены, что хотите напечатать этот документ?». Такие сомнения только приводят в ярость и идут вразрез с представлением об обходительном человекоподобном поведении.

Обходительная программа концентрируется на важном

Когда я заказываю салат в хорошем ресторане, мне подают хороший салат. В плохом ресторане меня начинают с пристрастием допрашивать: «Вы желаете „Цезарь“, салат со шпинатом или с овощами? Добавить лук? Сухарики? Тертый сыр? Пармезан или романо? Полная порция или к обеду? Соус французский, итальянский, масло, уксус или „Тысяча островов“? Соус подавать отдельно? Салат вынести до или после основного блюда?» Даже самого требовательного гурмана сервировка салата беспокоит не настолько, чтобы из-за этого подвергаться таким пыткам, тем не менее интерактивные системы проявляют подобное поведение постоянно. Так, программа Adobe Photoshop печально известна своими попытками закидывать пользователя тонной ненужных и раздражающих мелких вопросов в отдельном диалоговом окне каждый.

Необходительные программы имеют склонность задавать массу назойливых вопросов, большинство из которых совершенно бесполезны, а потому выбор какого-либо варианта ответа не приносит особых выгод пользователю, а только становится для него дополнительной головной болью.

Да и сам выбор тоже можно предоставить в разных формах. Варианты могут быть предложены, словно товары на витрине. Мы можем смотреть сквозь стекло в свое удовольствие, изучая товар, выбирая подходящий или вовсе игнорируя все предложения. Можно предложить их и по-другому: просто обрушить все варианты на покупателя со всей мощи, огорошив его, словно грозный таможенник на границе со своим вопросом: «*У вас есть что декларировать?*» Таможенник хорошо знает, что можно сколько угодно скрывать что-либо, однако страх быть пойманным обескураживает человека. Человек не знает, что стоит за этим вопросом. Будут ли его обыскивать или нет? Если он знает, что обыск неизбежен, он не станет лгать. Если же обыска не предвидится, он не устоит перед соблазном нелегально провезти лишний блок Marlboro.

Обходительная программа позволяет обойти правила

Когда механизмы ручной обработки информации заменяются автоматизированными компьютерными системами, процесс трансформации неизбежно происходит с потерями. Компьютеризация ручной обработки обычно нужна, чтобы увеличить объемы обрабатываемых данных, а не изменить функциональные возможности системы. Тем не менее системы ручной обработки обладают большей гибкостью, и не слишком просто определить, в чем именно эта гибкость заключается, чтобы превратить ее в отдельную функцию. Автоматизированная система работы с заявками способна обработать на несколько миллионов заявок больше, чем это сделает офисный сотрудник, однако этот сотрудник обладает особым умением – он способен *подчинить себе* всю ручную систему.

Стоит автоматизировать эту систему, как способность подчинить ее себе у человека пропадает. Повлиять на функционирование такой системы практически невозможно, а значит, получить некие преимущества не получится.

При ручном подходе, если бы офисному сотруднику позвонил знакомый из отдела продаж и сообщил, что нужно ускорить обработку заявки, потому что это увеличит количество заявок в будущем, офисный сотрудник смог бы на это повлиять. Или если бы поступила какая-то другая заявка с неполной важной информацией, офисный сотрудник все равно смог бы приступить к обработке, делая себе пометку, что нужно получить и заполнить недостающие сведения чуть позже. Компьютерные системы, как правило, такой гибкости лишены.

Компьютерная система может пребывать лишь в двух состояниях: фиксировать полное отсутствие или же полное соответствие информации заданным требованиям. Все прочие промежуточные состояния она не принимает и даже не способна распознать. В то время как в ручной системе обработки негласно существует еще одно весьма значимое состояние, которое нигде не записывается, но ощутимо присутствует в голове, на рабочем столе или в кармане человека-оператора, – это состояние *ожидания*.

Например, для того чтобы сформировать накладную, компьютерной системе требуется информация как о покупателе, так и о заказе. Если офисный сотрудник просто отправит заявку, не содержащую детальной информации о покупателе, в обработку, компьютерная система отклонит эту операцию, не желая выдавать накладную без нужных сведений.

Подобную способность человека действовать, отступая от принятой последовательности операций или до того, как все необходимые условия для начала совершения действия будут выполнены, я называю «отступлением от правил». Эту способность обычно приносят в жертву компьютеризации одной из первых. Из-за ее отсутствия цифровые системы становятся такими бесчеловечными, и такой результат – естественное следствие использования модели реализации. Программистам неясно, для чего создавать промежуточные состояния, ведь у компьютеров в них нет необходимости. Тем не менее для человека возможность внести незначительные изменения в систему очень важна.

Одним из самых больших достоинств системы с отступлением от правил является снижение количества ошибок. Если мы допускаем наличие в системе небольших, временных ошибок, доверяя человеку исправить их до возникновения необратимых последствий, мы имеем возможность избежать более крупных и серьезных проблем. Как ни парадоксально, большинство жестких ограничений в компьютерных системах устанавливаются именно для того, чтобы предотвратить возникновение небольших ошибок. Однако при таком подходе система лишается гибкости, превращаясь во врага человека. Пользователь не может вмешаться в ее работу и предотвратить серьезные последствия, а это приводит к тому, что человек и вовсе перестает беспокоиться о безопасности программы и избегания по-настоящему огромных проблем. Когда негибкие правила ставят ограничения для гибких пользователей, обе стороны оказываются в проигрышном положении. Жесткие рамки, ограничивающие действия человека, отрицательным образом сказываются на бизнесе в целом, а компьютерным системам все равно в конечном счете приходится иметь дело с недостоверными данными.

* * *

Способность отступать от правил – это одно из качеств обходительного человека, которое наиболее сложно реализовать в компьютерной системе. Такое качество требует куда более «восприимчивых» интерфейсов. Для воплощения этой способности в жизнь системе придется «обнажить» свои внутренние процессы пользователю со средними навыками взаимодействия с компьютером. Офисный сотрудник не сможет поместить заявку в самый верх стопки документов, если у него не будет возможности видеть величину всей стопки, ее границы и местоположение. Для перемещения заявки наверх электронной «стопки» понадобятся дополнительные инструменты, и инструменты эти должны быть хорошо видны – ровно так же, как и в системе с ручной обработкой, – тогда выполнить это действие будет так же легко, как переместить бумажный лист в настоящей стопке документов. С точки зрения реализации для механизма отступления от правил нужно приложить дополнительные усилия, чтобы организовать хранение заявок в состоянии ожидания, однако для реализации опции отмены операций требуется приблизительно то же самое. Самая большая проблема подобного подхода в том, что он открывает возможности для потенциального мошенничества и злоупотребления предоставленным доступом.

Обход правил системы можно истолковать как мошенничество, поскольку с технической стороны оно именно так и выглядит. При ручном подходе к обработке данных возможность обхода правил присутствует всегда, а потому на это закрывают глаза. Каждый раз подобная ситуация рассматривается как временное допущение, совершенно особый случай, и инициатор, разумеется, обработает все такие заявки до того, как отправится домой, в отпуск или на другую работу. Конечно же, перед различными аудиторскими проверками все эти «хвосты» подчищаются. В противном случае, если бы такая практика пренебрежения правилами была широко известна, это бы провоцировало сотрудников на совершение мошеннических действий.

Более того, если такой метод детально описать в документах компании, тем самым упрочив его положение, то некоторые сотрудники, обладающие не слишком сильной волей, могут увидеть в этом повод избегать делать свою работу вовремя и полностью или обманным путем заполучить деньги компании. Получается, что для компании поддерживать подобный обход правил – шаг не слишком финансово выгодный.

Тем не менее возможность отступления от правил меняет отношение к системе самих пользователей кардинальным образом. Все причины, по которым компании не следует добавлять в систему возможность обхода правил, звучат очень рационально и подтверждаются

логическими доводами (в том числе и со стороны законности действий). К сожалению, эти доводы строятся на основе идеализированного представления, а не реального отражения ситуации. Какую бы сферу бизнеса мы ни взяли – везде можно увидеть примеры, как сотрудники используют отступление от правил при ручной обработке данных; все это позволяет поддерживать плавное течение рабочих процессов – течение жизни. И потому невероятно важно, чтобы автоматизированные системы тоже обладали этим жизненно необходимым качеством, вне зависимости от возможных преград.

Положительный момент такого подхода с отступлением от правил применительно к автоматизированной системе заключается в том, что компьютер тоже может легко проверить действия пользователя и фиксировать их в деталях для любого внешнего наблюдателя. Это в разы перевешивает недостатки возможного злоупотребления. Здесь нужно следовать одному простому принципу: позвольте пользователю делать все, что ему хочется, но ведите подробный учет всех совершенных операций, чтобы при необходимости можно было обратиться к истории действий.

Обходительная программа поощряет незамедлительно

Программирование предполагает отсроченное удовольствие от результата. Компьютеры не могут ничего сделать сами, пока вы не приложите титанические усилия и первым делом не напишете программу для них. Этот принцип отложенного поощрения разработчики программного обеспечения усваивают весьма медленно, а потому программы, которые они пишут, работают по этому же принципу, вынуждая пользователя вводить всевозможные данные, прежде чем они смогут приступить к выполнению даже самой простой операции. Если бы таким образом вел себя человек, вы бы испытывали к нему неприязнь.

Наши программы стали бы намного более обходительными, если бы мы убедились, что они работают во благо пользователя и обеспечивают его всей информацией без лишней необходимости предварительно затрачивать массу усилий. Вспомним про телевизор моего друга Теда – у Теда должна быть возможность смотреть телепередачи и без настройки параметров устройства.

Обходительная программа вызывает доверие

Друзьям мы доверяем, когда на них можно положиться и когда они готовы жертвовать собой. Когда же компьютеры ведут себя необъяснимо и пренебрегают пользователями, ни о каком доверии речи не идет. Я испытываю доверие к оператору в кассе банка, потому что она одаривает меня улыбкой и помнит мое имя. А вот снимая наличные в банкомате, я всегда пересчитываю их, потому что полагаться на эту бестолковую машину я не могу.

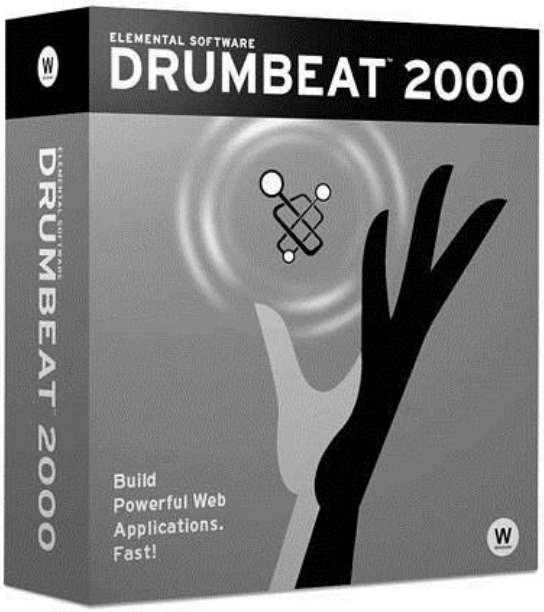
* * *

Мы испытываем неприязнь к программным продуктам вовсе не из-за их недостаточных функциональных возможностей, а из-за их необходительного поведения. Исходя из вышеизложенного списка характеристик, создать обходительную программу обычно не сложнее, чем раздражающую. Для этого всего лишь требуется вообразить, как посредством взаимодействия эмулировать такие качества, чтобы программа была похожа на сопереживающего и заботливого друга. При этом ни одна из указанных характеристик не конфликтует с другими целями, имеющими явную прагматичную направленность на задачи бизнеса. Обеспечить программу человечным поведением – это еще более прагматичная цель.

Кейс: программа Drumbeat от компании Elemental

Среди интересных проектов, в которых нам довелось поучаствовать, был проект от небольшого стартапа из Сан-Диего – компании *Elemental Software*. В качестве одного из своих программных продуктов они предлагали инструмент под названием Drumbeat, с помощью которого можно было создавать динамические веб-сайты с базой данных в основе.

Создать набор персон для работы над этим проектом было совершенно необходимо, хотя в нем в итоге оказалось всего два очень простых описания людей, у которых нет даже фамилий[29]. После создания персон и определения их целей к нам пришло понимание, которое впоследствии абсолютно перевернуло всю философию проектирования продукта для этой компании.



С самого начала нашего взаимодействия *Elemental* задала очень высокую планку. В планах компании было создать программу, в разы превосходящую по возможностям программные продукты конкурентов. Помимо этого, программа должна была отличаться легкостью в применении, также недостижимой конкурентами. Две эти цели вовсе не были несовместимы. Большая часть проблем, с которыми мы столкнулись, возникла по причине того, что программный продукт *Elemental* был ранее приобретен ими у другой компании и нам пришлось отталкиваться от этого – надстраивать новое поверх уже существующего кода. Из-за этого возникал постоянный конфликт между нашими задумками и тем, что уже было.

В существующем продукте было несколько довольно хороших функций, но его разработчики при создании руководствовались смутным представлением о пользователях. В результате пользоваться этими функциями было весьма затруднительно, а потому в целом продукт представлялся недостаточно мощным. Новый вице-президент, ответственный за разработку, – Эд Форман – возлагал на нашу компанию Cooper Interaction Design очень большие надежды. Так как он и сам недавно занял этот пост и еще не заручился доверием команды программистов, он рассчитывал, что с нашей помощью удастся радикально изменить взгляды на продукт. Как бы то ни было, Эд был превосходным лидером, предоставив нам время для общения с его командой, чтобы узнать друг друга получше и дать команде возможность познакомиться с нашими методами работы.

Исследование

Исследуя проблему, мы провели интервью с несколькими специалистами, преимущественно веб-мастерами. По ходу дела в наших головах стала вырисовываться все более ясная картина. Всех разработчиков веб-сайтов можно было четко разделить на два лагеря. Как и полагается, мы определили по одной персоне для каждого лагеря, и эти образы стали ключом к разгадке всей головоломки под названием Drumbeat, хотя это произошло не так, как мы ожидали.

Спустя всего несколько дней после старта мы уже могли грубо накидать портреты двух наших веб-разработчиков и дать им имена – Бетси и Эрни.

Бетси – дизайнер. Одевается во все черное и пьет эспрессо. До этого была художником, но после случайного «укуса» паука из Всемирной паутины стала создавать эскизы экранов вместо бумажных эскизов. Прочитала немалое количество книг и самостоятельно научилась создавать

симпатичные, но простые, статические веб-страницы. Изучила основы HTML, однако совершенно не разбирается – и не желает разбираться – в науке программирования. Собственный сайт Бетси представляет собой сочетание модного дизайна, сглаженных шрифтов, асимметричных вставок пастельного цвета и цитат из Патти Смит и Эстер Дайсон.

Каждый раз, когда Бетси требуется обработка данных на более продвинутом уровне, она обращается к Эрни. Эрни – помешанный на компьютерах программист, представитель нового поколения. Без ума от компьютеров, компьютерных игр, языков программирования и компьютерного оборудования. Пока еще не достиг весовой категории программистов старой закалки: не владеет такими языками, как C, C++ и ассемблер, но с невероятной легкостью использует инструменты наподобие CGI, Perl, JavaScript и Visual Basic. Знаком с сотней компонентов ActiveX и JavaBeans. Способен сотворить многофункциональный программный модуль из сложных компонентов всего лишь за несколько дней, в то время как программисту на C в далеких 1980-х на ту же задачу потребовалось бы около четырех лет. Собственный сайт Эрни представляет собой хаотичную подборку цитат из космической саги «Звездный путь» и «Симпсонов». Восемь разных шрифтов, кричащие красные буквы на черном фоне, мигающий текст, потоковое аудио, дергающиеся иконки, кнопки Submit и ссылки на самые лучшие сайты, посвященные игре Quake.

Мы очень быстро поняли, что у команды разработки *Elemental*, ввиду отсутствия ясного представления о потребностях Бетси и Эрни, было намерение осчастливить обе персоны. В результате у них получилась размытая мешанина мощных и сложных функций в графическом воплощении. «Только гляньте, какую невероятную вещь может теперь делать пользователь!» – восторгались они. Их «пользователь» был эластичным, и они совершенно не имели представления о его целях. Образ Бетси в целом вызывал симпатию у всех программистов в Elemental, однако по натуре им был ближе Эрни, вследствие чего продукт самым естественным образом подстраивался именно под его потребности.

После того как мы представили персоны Бетси и Эрни, вся компания моментально увидела в них чрезвычайно знакомые архетипы и смогла полагаться на них как на полезные описания характеров пользователей.

Кто от кого зависит

Рынок инструментов для визуального создания веб-сайтов всегда был (и остается) разогретым, так что на нем присутствовало множество конкурентов, но теперь у нашего клиента впервые появилась возможность посмотреть на свой продукт и продукты конкурирующих компаний взглядом Бетси и Эрни.

Конкурентный рынок разделился надвое по линии «Бетси – Эрни». Компании, стоявшие по одну сторону, разрабатывали новые великолепные инструменты исключительно для Эрни. Эти продукты были многофункциональными и непростыми в использовании, однако с их помощью Эрни мог создавать серьезные, мощные динамические веб-сайты для корпоративных клиентов.

По другую сторону стояли компании, которые занимались разработкой новых великолепных инструментов исключительно для Бетси. Эти продукты были простыми в использовании, визуальными и несложными в освоении, но слабыми, как котята. С помощью них можно было создавать только статические веб-сайты с небольшим функционалом, в которых напрочь отсутствовала возможность подключать внешние базы данных.

Когда мы оценили всю ситуацию сквозь призму «Бетси – Эрни», нам всем стало совершенно ясно, что хорошим шагом здесь будет создать такой инструмент для Бетси, который расширит ее собственные возможности. Это даст Elemental шанс разработать желанный продукт для еще не занятой рыночной ниши. Вскоре после этого имя «Бетси» превратилось в боевой клич для программистов компании – они сконцентрировали все усилия на том, чтобы помочь ей.

Старт был взят довольно бодро, однако чуть дальше в ходе проектирования мы изучили цели Бетси более пристально и выявили одну интересную вещь.

В старые времена первого поколения веб-сайтов, когда они были простыми и статическими, Бетси выполняла свою работу независимо. У нее было все, чтобы отрисовать дизайн сайта и создать собственно сайт для клиента без необходимости обращаться за помощью к Эрни. Так как Бетси делала то, в чем сама была экспертом, она могла сориентировать потенциального клиента по объемам работ, срокам и стоимости проекта. И она могла с уверенностью полагаться на

собственные обещания. Именно такое самоопределение и возможность быть независимой и привлекли внимание Бетси к Всемирной паутине в первую очередь. Именно это побудило ее оставить основную работу и начать работать на себя.

С ростом интернета расширялись и его возможности, но вместе с тем увеличивалась и сложность создания веб-сайтов. Сайты становились динамическими, их функциональность расширялась, все большее количество из них обращалось к базам данных напрямую. Теперь для Бетси уже было затруднительно работать с такими низкоуровневыми концепциями из области программирования. Более того, сфера программирования ее совсем не привлекала, она *не хотела*проходить дополнительное обучение. Тогда-то на ее пути возник Эрни, способный справиться со всеми техническими сложностями за нее. Эрни все эти программистские штуки просто обожал.

Однако вскоре Бетси осознала, что теперь зависит от Эрни во всех аспектах касательно предоставления выполненного заказа клиенту. В процессе создания каждого нового веб-сайта она неизбежно доходила до того момента, когда требовалось призвать на помощь Эрни, чтобы он настроил доступ к базе данных и написал код для динамических веб-страниц. Теперь Бетси уже не могла полагаться полностью только на себя и создавать полноценные сайты самостоятельно, без участия Эрни. А Эрни, в свою очередь, был не столь пунктуальным, как Бетси. Бетси больше не могла гарантировать клиенту, что проект будет выполнен к конкретному сроку. Такая несобранность, свойственная Эрни, тормозила бизнес Бетси. Все это позволило нам взглянуть на цели Бетси под несколько иным углом.

Несмотря на то что Бетси все еще хотела заниматься созданием прекрасных, мощных, динамических веб-сайтов, ее основная цель изменилась. С исчезновением независимости Бетси, что ранее воспринималось как данность и как цель «гигиеническая», эта потребность вышла на первый план. Все внимание Бетси теперь было направлено на желание освободиться от необходимости взаимодействовать с Эрни. Она хотела выстраивать доверительные отношения с клиентом и проектировать, а затем создавать красивые, мощные, динамические веб-сайты с базой данных в основе, *без необходимости ждать, пока Эрни раскусит очередную техническую загадку*.

Наше изначальное видение подразумевало, что мы сделаем более мощный инструментарий для Бетси, сохраняя легкость в его использовании. Тем не менее такой продукт при всей его желанности просто давал бы Бетси отсрочку по времени, когда бы ей ни приходилось обращаться к нему за помощью, но ее основная цель при этом оставалась бы недостигнутой. Чтобы обеспечить успех Бетси, нам нужно было спроектировать продукт Drumbeat таким образом, чтобы он позволял ей завершать все проекты самостоятельно.

Для Эрни работа с Бетси тоже была не самым приятным времяпровождением на свете. На каждое его действие требовалось одобрение Бетси, а кроме того, она беспрестанно придиралась к нему то по поводу пикселя тут, то по поводу пикселя там, хотя с его точки зрения все это было таким незначительным. Она настаивала на переделках уже готовой работы по пять-шесть раз, заставляла вносить ненужные, на взгляд Эрни, правки до тех пор, пока не была полностью довольна. Эрни хотел освободиться от влияния Бетси не меньше, чем она хотела освободиться от него.

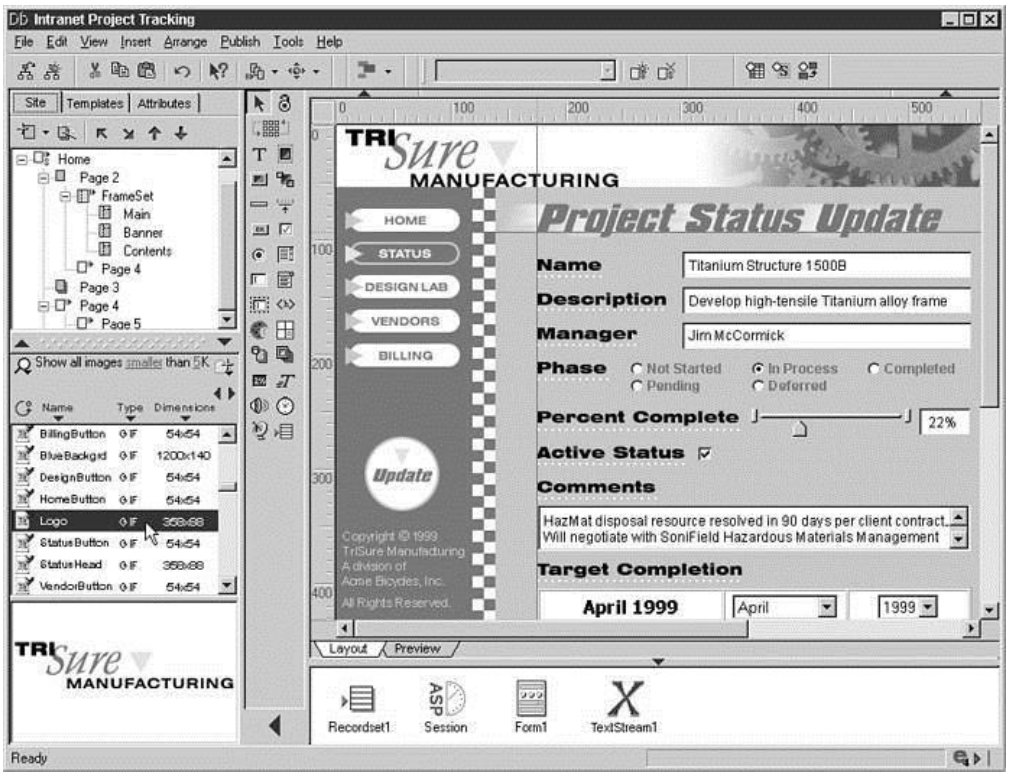
Проектирование

Теперь мы могли наиболее точно и понятно описать ситуацию применения этого программного продукта. Взамен того, чтобы предоставлять Бетси отсрочку во взаимодействии с Эрни, нам нужно было воздвигнуть между ними непроницаемый барьер, гарантируя независимость для обоих. Бетси все еще нуждалась в функционале, который мог разработать Эрни, и в конечном счете Бетси оставалась неплохим источником новых проектов и доходов для Эрни, а потому их взаимодействие в коммерческом плане также требовалось сохранить, а вот задачи, наоборот, полностью разграничить.

На основании этих выводов мы заключили, что барьер между Бетси и Эрни следовало строить на основе общего стандарта – интерфейса, с помощью которого можно было бы создавать и задействовать функциональные модули. Мы решили дать Эрни доступ к интерфейсу программиста, где он мог бы писать код, связанный с другими компонентами сайта, а Бетси – доступ к интерфейсу дизайнера, где она могла создавать проекты целиком. Весь программный

продукт Drumbeat в этом случае стал бы для них единой нейтральной территорией. У Эрни появлялась возможность писать мощные, гибкие модули и публиковать их, используя функциональный интерфейс Drumbeat. Бетси же сможет воспользоваться этими модулями посредством визуального программного интерфейса продукта.

У самой же Бетси появлялась возможность создавать динамические веб-сайты с базой данных в основе, используя готовые модули и при этом без необходимости взаимодействовать с их автором. Эрни в этом случае будет писать, публиковать и продавать функциональный код, и ему не потребуется когда-либо менять цвет фона сайта. Сделав обе персоны независимыми друг от друга, мы наделили Бетси более широким инструментарием для дизайна и производства, а Эрни – функциональными возможностями для программирования.



В соответствии с такой постановкой задачи Эрни выступает уже не в роли программиста, пишущего код на заказ, а в роли автора новых инструментов. Теперь он может создавать совместимые подключаемые модули, которые Бетси при необходимости может удобно подключать к своим инструментам. Применение его модулей значительно расширяется, так как ими могут воспользоваться и другие «Бетси», чтобы применять их в собственных сайтах.

Этот кейс интересен тем, что в данном случае проектирование взаимодействия значительным образом повлияло как на внутреннее устройство программного продукта, так и на его позиционирование на рынке. Это отличный пример того, как проектирование способно воздействовать на внутреннее через описание только лишь внешнего.

Внезапное препятствие

Поначалу разработчики компании *Elemental* неохотно приняли предложенное нами решение. Они пытались убедить нас, что оно не сработает, так как им в голову сразу пришло несколько исключительных ситуаций, когда Бетси все же мог понадобиться особый талант Эрни. «Мы не можем вовсе исключить Эрни из процесса, – заявили они, – ведь у Бетси может возникнуть необходимость сделать нечто специфическое или особо сложное».

Что ж, мы согласились с тем, что подобное может случиться, но вероятность крайне мала. В большинстве случаев она будет работать независимо, в то время как в текущем положении она *никогда* не бывает независимой. При наступлении описанных исключительных ситуаций ей просто придется на время снова становиться зависимой от Эрни. Это не сделает ее положение хуже, при этом в большинстве случаев положение окажется заметно лучше.

По причине того, что Бетси ценит собственную независимость крайне высоко, она согласится на разумные жертвы ради ее достижения. Благодаря тому что с помощью Drumbeat она сможет создавать веб-сайты самостоятельно от начала и до конца, не прибегая к общению с Эрни, она готова терпеть небольшие компромиссы в дизайне, извлекая выгоду из готовых модулей Эрни[30]. Жертва оказывается несущественной, ведь весьма небольшой процент клиентов просит создать что-то, что не реализуется типичными шаблонами. Разумеется, если ей когда-нибудь доведется получить запрос от сети универмагов WalMart на создание внутрикорпоративного портала или от отелей Hilton на разработку системы быстрого бронирования номеров, ей придется обратиться к специалисту, разбирающемуся в сложном программировании, чтобы решить эти трудоемкие задачи, однако в большинстве случаев в этом нет необходимости.

Прочие вопросы

В первоначальной версии программы присутствовало множество плавающих панелей с инструментами для рисования, каждая из которых частично заслоняла область, где конструировался сайт. Участники команды Elemental по какой-то причине считали, что пользователям действительно *нравится* двигать эти панели по экрану в ходе работы. На каждой демонстрации программного продукта они гордо показывали эти панели.

Участники же нашей проектировочной команды были убеждены, что эти панели только раздражают пользователя, сбивают его с толку и на самом деле совершенно излишни. Конечно, доступ к инструментарию необходим, но нам были известны более удобные способы реализовать это. Тем не менее стоило нам негативно высказаться о панелях, как все программисты (равно как и руководители разработки) в один голос заявляли, что пользователи активно используют их.

Когда мы начали проводить эксперименты, наблюдая за тем, как продукт используют настоящие «Бетси», нам стало понятно, почему эти плавающие панели были столь популярны. В изначальном интерфейсе программы без этих панелей было не обойтись. При этом большинство инструментов из каждой панели использовалось крайне редко, но как минимум по два инструмента на панели были востребованными и полезными. Таким образом, Бетси нужны были все панели даже для выполнения самой простой задачи. Размер каждой панели был слишком велик из-за большого количества мало востребованных инструментов, все эти панели размещались поверх области, в которой непосредственно конструировался сайт, а потому Бетси постоянно приходилось убирать их из поля зрения, двигая по экрану. В качестве альтернативы Бетси могла закрепить все панели вдоль одной из сторон окна программы, однако это приводило к тому, что ей приходилось постоянно прокручивать макет сайта, чтобы добраться до нужной его части. Бетси оказалась между молотом и наковальней. Ей приходилось тратить массу лишнего времени либо на прокрутку сайта, либо на перемещение панелей. Подобные вынужденные и ненужные пользователю действия мы называем «акцизами». Первоначальная версия программы была такими «акцизами» переполнена.

Мы знали, что хорошим решением этой проблемы будет оставить под рукой только те инструменты, которые используются наиболее часто, и кроме того, объединить их в одну категорию. Если они будут разбросаны по экрану, Бетси запутается.

Применив прием несложной реорганизации инструментов на панелях, мы значительно уменьшили их объем, оставив только самые часто используемые функции. Затем мы закрепили все панели на экране в определенных позициях. Теперь они стали практически незаметны в интерфейсе. Это отличный пример того, как целеориентированное проектирование фактически уменьшило объем кода, который требуется для описания интерфейса.

* * *

Наш вариант проектирования, равно как и продукт, возымел успех. Когда реализация продукта, основанная на нашем проекте, почти близилась к завершению, компании *Elemental* удалось привлечь значительные венчурные инвестиции, отчасти благодаря инновациям в его пользовательском взаимодействии. С момента выхода программа Drumbeat удостоилась многочисленных хвалебных отзывов в отраслевых изданиях. Цитата из *PC Magazine* дает представление об этой ситуации:

«Продукт Drumbeat поистине впечатляет своей уникальностью. Он автоматизирует процессы создания сложных веб-сайтов и превосходит по своим возможностям все прочие решения на рынке. С помощью него любой человек, не знакомый с программированием, может с легкостью создавать сайты посредством drag-and-drop. Вы сможете создать серьезный веб-сайт на уровне профессионала, при желании используя технологию Active Server Pages, и при этом не напишете ни единой строки кода».

Продукт произвел фурор на рынке программного обеспечения, несмотря на тот факт, что многие другие продукты для создания веб-сайтов появились на рынке гораздо раньше.

* * *

Как можно было увидеть, такой взгляд на продукт сквозь призму целей пользователя открывает перед нами уникальные и мощные перспективы, предоставляя грандиозные возможности для применения творчества. Это и есть основная задача проектирования, ориентированного на цели.

11. Проектируем для людей

В предыдущих главах мы обсудили концепцию персон и убедились в важности преобладания целей над задачами. Только после того, как мы определили пользовательских персон и выяснили их цели, мы можем переходить к изучению задач, будучи уверенными, что они не нарушат процесс проектирования. Наш метод для объединения задач в единое целое мы называем «сценариями». Сценарий – это краткое описание действий пользователя, которые он предпринимает при взаимодействии с программным продуктом, пытаясь достигнуть своих целей. Далее в этой главе я остановлюсь на сценариях более подробно, а также расскажу о некоторых других инструментах, полезных при проектировании. Я также опишу кейс, из которого можно будет понять, как эти инструменты, в частности сценарии, работают на реальных проектах.

Сценарии

По мере того как в ходе процесса проектирования внимание смещается от общих концепций к проработке деталей, эффективность сценариев также повышается. Мы разыгрываем эти сценарии с нашими персонами, будто с актерами, изучающими свою роль, и через это проверяем правомерность наших предположений в отношении проектирования. Ничего удивительного, что такой процесс работы с нашими сценариями напоминает подход актера к постижению характера своего героя, когда актер вживается в образ персонажа, чтобы увидеть мир его глазами и ощутить его эмоции. Мы тоже пытаемся думать так, как это сделала бы наша персона. Мы на время забываем о собственном уровне образования, способностях, степени подготовки и методах, которыми владеем, воображая себя человеком, обладающим опытом и восприятием, присущим *персоне*. Учитывая, что мы являемся проектировщиками, а не актерами, воплотить подобное без понимания контекста и знания нюансов бывает нелегко, потому сценарии оказываются мощным подспорьем. К примеру, если мы обладаем информацией о том, что Бетси создает веб-сайт для страховой компании, нам становится гораздо проще проникнуть в самую ее суть. Это может показаться странным, однако на самом деле это вполне нормальная ситуация. Это так же, как программисты проникают в самую суть компьютеров. Для программистов становится привычным даже описывать действия компьютеров от первого лица, например, от них можно услышать: «Я получил доступ к базе данных и теперь сохраняю записи в своем кэше». Хотя они употребляют местоимение «я», не они выполняют все действия, а компьютер. Однако, ставя себя на место компьютера, им легче понять, что происходит у него внутри и что ему требуется, а затем реализовать это в коде.

Сценарии обычно составляются исходя из информации, полученной на первом этапе проведения исследования. Из интервью с пользователями, а также в ходе прямого наблюдения за их действиями о задачах можно узнать довольно много. Целям свойственна стабильность, они практически не меняются, а вот задачи изменчивы, и нередко при компьютеризации деятельности часть этих задач отпадает. Проходя этап разработки сценариев, мы традиционно отыскиваем задачи, которые существуют лишь потому, что так исторически сложилось, а затем, по мере возможности, избавляемся от них.

Качественный сценарий в большинстве случаев является таковым благодаря ширине охвата, нежели глубине детализации. Другими словами, более важно, чтобы в сценарии присутствовало описание какого-либо процесса от начала и до конца, чем подробное описание каждого шага с мельчайшими нюансами.

Здесь важно прорабатывать только те сценарии, которые будут способствовать процессу проектирования, и не углубляться в исключения. Обычно мы создаем сценарии двух видов, хотя их может быть и больше, – это сценарии на основе ежедневных действий и сценарии на основе обязательных действий.

Сценарии ежедневных действий

Сценарии ежедневных действий представляются наиболее важными и полезными. В них описываются самые частые действия пользователя. Например, для систем отслеживания ошибок и неисправностей в программах типичным сценарием, выполняемым ежедневно, будет, соответственно, поиск ошибок и заполнение отчетов об ошибках, найденных в текущем периоде. Любой специалист технической поддержки выполняет две эти задачи по несколько раз каждый день.

В целом у большинства пользователей количество таких описанных ежедневных сценариев будет весьма ограниченным. Обычно число таких сценариев не превышает одного или двух. Больше трех их бывает крайне редко.

Сопровождение сценариев ежедневных действий в пользовательском взаимодействии должно быть невероятно качественным. Новым пользователям требуется осваивать такие сценарии очень быстро, а потому нужна хорошая встроенная система обучения. Это значит, что пояснения к применению различных функций должны быть описаны прямо в интерфейсе самой программы. Тем не менее чем чаще пользователь работает с программой, тем меньше ему в дальнейшем требуется использовать эти пояснения и инструкции. Вместо этого вскоре возникнет потребность в освоении быстрых сочетаний клавиш. Более того, еще чуть позже, по мере приобретения опыта работы с программой, у пользователей возникнет и потребность в индивидуальной подстройке программы под их персональный стиль действий с учетом личных предпочтений.

Сценарии обязательных действий

В сценариях обязательных действий описываются все действия пользователей, которые нужно исполнять непреложно, однако нечасто. К этому виду сценариев можно отнести такие действия, как чистка баз данных и создание исключительных запросов. Взаимодействия на основе обязательных действий также требуют качественного обучения. Тем не менее в данном случае пользователь никогда не перерастет их и не станет применять параллельные действия вроде сочетаний клавиш. Ввиду нечастого использования этих операций пользователь подстроится под настройки программы и не потребует персонализации. Таким образом, разработчики будут освобождены от необходимости дорабатывать этот вид взаимодействия до того уровня, который требуется при использовании сценариев ежедневных действий. Приблизительно по той же причине салон нового «Ягуара» отделан со всей роскошью, в сравнении с грубой металлической окантовкой его моторного отсека.

Несмотря на то что почти у всех программных продуктов количество сценариев обязательных действий невелико, их количество все же превышает число ежедневных сценариев.

Сценарии исключений

Конечно же, есть и третий вид сценариев – сценарии исключений. Программисты по природе своей уделяют внимание исключительным ситуациям, тем не менее при проектировании такими сценариями можно пренебрегать. Это не значит, что соответствующий функционал можно удалить из программы, – это значит, что проектировать взаимодействие по этим сценариям можно грубо и прятать его поглубже в интерфейс. От способности обрабатывать исключения зависит лишь качество кода, а вот успех самого *программного продукта* обусловливается способностью справиться с ситуациями, описанными в сценариях ежедневных и обязательных действий.

Если пользователь выполняет какую-то задачу постоянно, соответствующее взаимодействие должно быть спроектировано на высшем уровне качества. То же справедливо и для задачи, которая выполняется редко, но неизбежно. Взаимодействие для нее хотя и преследует иные цели, также должно быть качественно спроектировано. А вот проектирование взаимодействия для

задач, выполнение которых необязательно или происходит не каждый день, не требует такого скрупулезного подхода. Ресурсы в виде времени и денег всегда ограниченны, а потому это отличная возможность сэкономить и перераспределить ресурсы на более полезные вещи. Готовиться нужно ко всем видам сценариев, но детально проектировать взаимодействие следует под те из них, которые наиболее важны или происходят чаще остальных.

* * *

Персоны, цели и сценарии – тяжелая артиллерия в нашем арсенале проектирования. Прежде чем переходить к изучению кейса практического применения сценариев, я бы хотел рассказать еще о нескольких полезных концепциях проектирования, таких как адаптивный интерфейс, вечная середина, терминология, мозговой штурм и латеральное мышление.

Адаптивный интерфейс

Взаимодействие всегда можно сделать еще проще – для этого достаточно просто удалить часть опций, уменьшив тем самым общую функциональность продукта. Нередко такая тактика не оправдывает себя, однако бывают и исключения. Задача проектирования более сложного уровня требует простоты в использовании продукта без принесения в жертву его функционала и производительности. Добиться подобного непросто, но вполне возможно. Для этой ситуации потребуется применить технику, которую я называю *адаптивным интерфейсом*.

Несмотря на то что в программе должно быть множество самых разных функций, у конкретного пользователя в каждый конкретный момент времени нет потребности использовать их все одновременно. Под каждый сценарий персоне пользователя понадобится использовать лишь небольшое подмножество элементов управления и связанных с ним данных, хотя это подмножество каждый раз может быть разным в зависимости от выполняемой задачи. Интерфейс станет в разы легче для восприятия, если нужные для реализации ежедневных сценариев элементы управления и данные разместить на видном месте, в то время как все прочие элементы будут перемещены на второй план, за пределы зоны видимости.

Интерфейсы многих серьезных программ схожи с меню в китайском ресторане, где каждая страница испещрена сотней вариантов выбора. Это может казаться подходящим при заказе ужина, но вот в высокотехнологичных продуктах такая особенность крайне мешает восприятию.

Например, на стандартной панели инструментов Microsoft Word расположены значки загрузки, закрытия и печати документа. Пользователь обращается к этим задачам довольно часто, поэтому такое размещение вполне оправданно. Однако тут же рядом с ними можно обнаружить и значки для генерации схемы документа и внедрения электронных таблиц. Компания *Microsoft* разместила значки этих опций на основной панели инструментов, чтобы мы смогли оценить всю мощь программы. Только, к несчастью, большинству пользователей эти опции не пригодятся никогда, а даже если и пригодятся, они не будут пользоваться ими постоянно. Этих функций не должно быть на панели инструментов, потому что панель инструментов – это компонент интерфейса, традиционно предназначенный только для самых востребованных функций.

Вечная середина

Как правило, самые мощные из наших инструментов проектирования позволяют нам лучше понять личности наших пользователей, представить, как они выглядят, и проникнуть в самую их суть. Одна ментальная модель, которую мы применяем постоянно, называется «вечная середина». Большинство пользователей нельзя отнести ни к неопытным, ни к экспертам – они находятся где-то посередине. Вспомните Рупака, Шэннон, Декстера и Роберто, об уровне компьютерной грамотности которых шла речь в главе 9 «Проектируем для удовольствия». Несмотря на то что их опыт и специализация весьма различны, всех их можно отнести к категории вечной середины.

Опыт людей, которые взаимодействуют с интерактивными системами, можно изобразить с помощью кривой нормального распределения (колоколообразной кривой). Если мы построим такой график для любого электронного продукта, изобразив по вертикали количество пользователей, а по горизонтали их уровень компьютерной грамотности, мы увидим, что в крайней левой и крайней правой позиции будет небольшое число неопытных и продвинутых

пользователей соответственно, а преобладающее количество средних значений займет всю центральную часть.

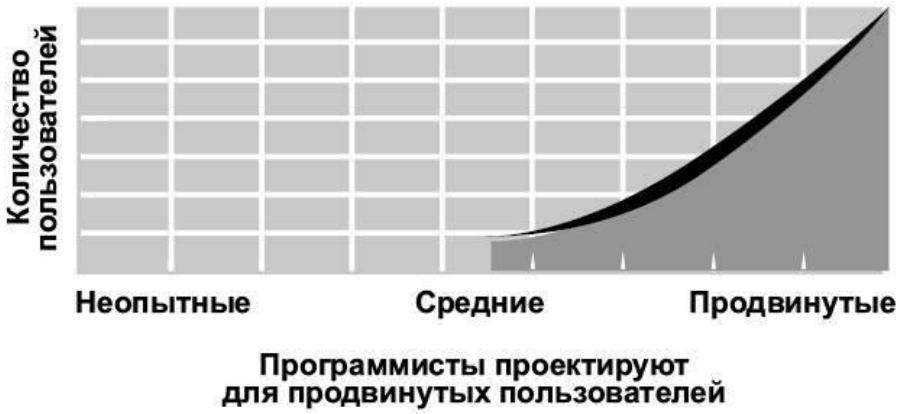
Однако статистика не дает полного представления о происходящем. Это лишь моментальный снимок, не меняющийся во времени. И хотя большинство людей со средним уровнем компьютерной грамотности остаются в этой категории довольно продолжительное время, люди из крайних положений этой кривой – неопытные и продвинутые – меняются постоянно. Сохранять высокую степень экспертности сложно, а потому продвинутые пользователи быстро перестают быть таковыми, а неопытные меняются с еще большей скоростью.

Несмотря на то что каждый пользователь начинает с уровня неопытного, никто не пребывает в этом статусе долго. Так происходит оттого, что никому не нравится быть начинающим, это никогда не является чьей-либо целью. Люди не любят чувствовать себя некомпетентными, а неопытные пользователи по определению некомпетентны. В противоположность этому, обучение и развитие собственных навыков – процесс вполне естественный, приносящий немало удовольствия и радости, а потому неопытные очень быстро дорастают до уровня средних пользователей. Так, обучение игре в теннис – занятие весьма увлекательное, но эти первые несколько часов или дней, когда вы пропускаете мячи, а они вылетают за пределы поля, могут расстроить кого угодно. Но стоит вам овладеть базовыми навыками обращения с ракеткой, как вы уже не будете тратить так много времени, бегая за пропущенными мячами, а ваш уровень значительно повысится. Быть новобранцем не так уж приятно, а потому каждый старается побыстрее преодолеть это состояние, дотягиваясь до некоего подобия среднего. Если же спустя несколько дней вы все еще будете хаотично колотить по мячу, запуская их по всему теннисному корту, вы просто-напросто вскоре оставите это занятие и попробуете себя в рыбной ловле или коллекционировании почтовых марок.

Неопытные пользователи, отображенные в левой части кривой, обычно перемещаются в центральную часть либо вообще пропадают с графика, так как обзаводятся новым занятием, в котором их *способности* дорасти до среднего уровня значительно выше. Центральная же часть графика характеризуется невероятной стабильностью. Достигнув приемлемого уровня, пользователи обычно так и пребывают на нем постоянно. Наиболее справедлива такая ситуация в отношении продуктов с высоким коэффициентом когнитивного сопротивления, поскольку радости от изучения таких продуктов пользователи не получают. Они осваивают лишь минимум самых необходимых функций, и далее их прогресс останавливается. От освоения сложных систем способен получить удовольствие лишь представитель вида *Хомо логикус*.

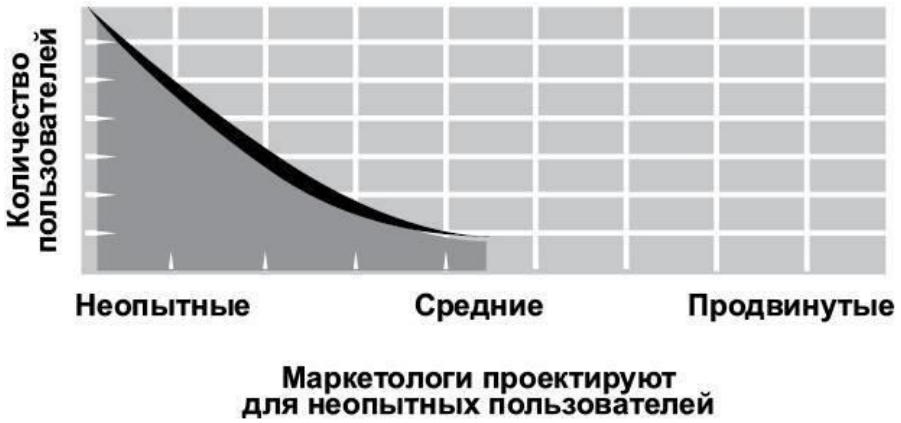
Теперь сравним эту колоколообразную кривую с графиком, который описывает разработку программного обеспечения. Программисты являются продвинутыми пользователями, так как они изучают абсолютно все, в том числе и самые неожиданные и исключительные ситуации с невысокими шансами на возникновение, чтобы затем реализовать в коде их обработку. А поскольку они от природы склонны проектировать, принимая во внимание лишь собственные предпочтения, мы можем сделать вывод, что код, который они создают, отражает модель реализации, где приоритет всех функций в пользовательском взаимодействии одинаков. Отобразив на графике кривую пригодности типичного продукта, созданного по модели

реализации, мы увидим, что пиковая точка придется на крайнюю правую часть шкалы, в зоне обитания продвинутых пользователей. До пользователей среднего уровня никому особо дела нет.



Программисты проектируют для продвинутых пользователей

Внутри компаний происходит следующее: продавцы, маркетологи и руководители постоянно демонстрируют продукт покупателям, прессе, партнерам и инвесторам, не знакомым с этим продуктом ранее. Получается, что профессионалы отрасли ввиду производственной необходимости вынуждены постоянно взаимодействовать с неопытными пользователями. Это накладывает отпечаток на их восприятие всего сообщества пользователей через призму ограниченной проблемной группы. Потому упомянутые профессионалы, обладающие определенным влиянием, настаивают на создании более простых интерфейсов в угоду неопытным пользователям. Они также хотят встроить в продукт обучение, которое бы облегчило участь новичков. Теперь пиковая точка кривой нашего графика располагается в крайней левой части шкалы, в зоне обитания неопытных пользователей.



Маркетологи проектируют для неопытных пользователей

Наложив один график на другой, мы увидим: помимо того, что два этих подхода оказывают на проектирование взаимодействия сильнейшее противоположное влияние, эти графики еще и не совпадают с главной картиной. Программисты настаивают на взаимодействии, подходящем лишь для таких же продвинутых пользователей, в то время как маркетологи требуют взаимодействия для новичков. Самой же обширной, неизменной и самой важной группой пользователей – со средними навыками – попросту пренебрегают.

Парадоксальные кривые

Такое несоответствие между тем, как видят пользователей разработчики, и тем, какими они являются на самом деле, влечет увеличение коэффициента когнитивного сопротивления. Такую ситуацию легко увидеть на практике в большинстве внутрикорпоративных программ и среди массовых программных продуктов. Для успешного применения этих продуктов нужно самому быть программистом, однако при этом подобные продукты «набиты» всевозможными артефактами вроде мастеров и сетевой справочной системы для новичков. Такие возможности являются яркими представителями наспех «приваренных» вспомогательных обучающих опций. Мастеры и справочная система, конечно, помогают пользователям в определенных ситуациях, однако не обучают их тому, как избежать подобных случаев в будущем. Продвинутые пользователи никогда не прибегают к использованию справки и мастеров, а новички стараются поскорее избавиться от этих уничижительных напоминаний об их невежестве. А вот представители вечной середины навсегда остаются опутаны узами этих артефактов.

* * *

Пополнив наш арсенал целеориентированного проектирования важными инструментами персон, целей, сценариев, вечной середины, адаптивного интерфейса и некоторыми другими, мы можем с уверенностью преодолевать проблемы клиентов. Нам известно, что в конечном счете даже самая сложная задача падет под напором нашего подхода к проектированию.

Упражнение «Волшебный компьютер»

Каждый инженер обладает собственным видением продукта, но из-за того, что он занимается исключительно программированием, он крайне редко способен посмотреть на продукт взглядом конкретного пользователя (по крайней мере, не с той позиции, где я нахожу это полезным). Во время проводимых нами сессий мозговых штурмов мы избавляемся от различных ограничений и ожиданий, начиная проектировать с чистого листа, но с самым пристальным вниманием к персонам и их целям. Мы часто применяем упражнение на включение креативного мышления, которое начинается с фразы «Представь себе, что у нас есть волшебный компьютер…», смысл которой заключается в том, чтобы разыграть сценарий действий пользователя, применяя «волшебный компьютер», у которого нет никаких ограничений.

В ходе этого упражнения становятся ясно видны различия между целями и задачами. Задачи склонны изменяться вместе со сменой технологии, а вот цели остаются неизменны. Применяя хитрость с волшебной технологией, мы заставляем все задачи измениться, после чего на первый план выходят настоящие цели. Невзирая на то что все концепции являются воображаемыми, сам процесс представляет собой совершенно простое ментальное упражнение. Иногда верный ответ возникает в голове проектировщиков как озарение, но с равной степенью вероятности он является результатом продолжительных дискуссий и изучения проблемной области.

Терминология

В ходе проектирования и в особенности в ходе сессий мозговых штурмов я делаю отдельный акцент на необходимости создания и использования подробной и точной терминологии. Я

разделяю мнение, что технические нюансы проектирования интерактивных продуктов настолько важны, что даже одно неверно истолкованное слово может сорвать целый проект. Я лично был свидетелем того, как разными участниками команды клиента употреблялись широко известные слова наподобие «кнопка» или «диалоговое окно», и при этом в совершенно разных значениях. Помню одну из встреч с заказчиком, на которой десять высококлассных профессионалов в течение двух часов вели ожесточенную полемику по поводу разногласия, возникшего вследствие того, что разные участники употребляли разные термины для одних и тех же концепций.

Если для выражения какой-либо идеи нет подходящих слов, передать ее становится практически невозможно. Как следствие, невозможно произвести анализ и декомпозицию этой идеи на таком уровне технической детализации, чтобы можно было описать ее на языке C# или Java.

Когда ясных слов нет, программисты рефлекторно обращаются к единственному доступному им и наиболее точному способу выражения мысли: исходному коду. Невзирая на то что точнее кода и быть ничего не может, это средство выразительности также менее всего подвластно изменениям. Потому подобная неразбериха с терминами нередко заставляет программистов приступать к написанию кода слишком преждевременно, отчего такой код становится начальной точкой отсчета при проектировании, вне зависимости от его правомерности.

Если объем терминологии неполон или понятия лишены четкости, мышление людей следует традиционным концепциям. Без надежных и точных терминов описывать новые идеи становится затруднительно, а потому от них отказываются преждевременно.

Под терминологией я понимаю вовсе не те фразы, что будут напечатаны на внешней упаковке продукта. Эти термины используются только для общения внутри команды, поэтому нас не беспокоит правильность фраз на взгляд маркетолога. Единственное требование: они должны точно отражать суть. Впоследствии маркетологи подберут другие слова, которые больше подходят для отражения потребительских свойств продукта. Например, продукт ScanBank от компании Logitech первоначально прозвали «Сдающим», и это описание вполне подходило под наш процесс проектирования, но широкой публике о нем ничего не было известно.

В другом проекте наша проектировочная команда намертво застопорилась с одной проблемой. В ходе многочисленных дискуссий стало понятно, что кто-то из нашей команды использует термины иначе, чем остальные. Неэффективность наших обсуждений стала следствием отсутствия единой терминологической базы. Я убедил всех в необходимости разбить проект на более мелкие фрагменты, удобные каждому из нас, и назвать их совершенно новыми именами, без всякой связи с их сущностью. Без особых на то причин я обозначил их через названия горных цепей Аляски. Четыре основных блока продукта получили названия «Святой Илья», «Брукс», «Аляска» и «Врангель». Сначала мы вдоволь посмеялись над столь неуместными обозначениями, но затем почти моментально пришли к консенсусу и дальнейший прогресс проекта пошел значительно быстрее.

Коммуникационный прорыв

Первая задача надежной терминологии – сделать коммуникации в команде более эффективными. Тем не менее нередко у нее бывает и еще одна немаловажная задача. Время от времени мы замечаем, что в корпоративной культуре компаний наших клиентов уже прижились отдельные термины. Хорошим примером здесь будет фраза *Microsoft* «Embrace the Internet» («Охвати весь интернет»). Такие фразы порой обретают почти религиозный смысл, их произносят с благоговением. Из-за этого благоговейного трепета добраться до первоначальной сути фразы и рассмотреть ее в свете новых императивов проектирования бывает очень нелегко. Следует ли из фразы, что под охватом интернета подразумеваются браузеры, или имеется в виду HTML, или же только протокол TCP/IP? Подобные «священные слова» похожи на ограждение вокруг храма. Однако попирать священную веру клиента мы тоже не можем, поскольку это не продвинет наш процесс проектирования вперед. Поэтому мы обычно разбиваем процессы, задачи и программы на понятные отдельные фрагменты и называем их по-новому, названиями, которые не несут прежнего смысла. Новые названия, как правило, также носят некий юмористический оттенок, и с помощью такого легкомысленного подхода нам удается прорваться сквозь серьезный настрой участников.

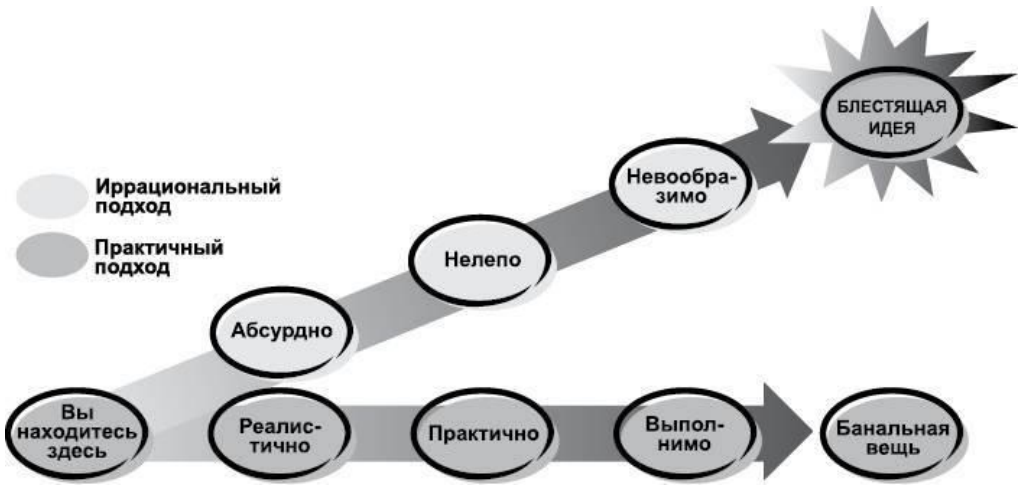
Последнее слово за реальностью

Классический процесс разработки программного продукта начинается с выявления пределов и ограничений. Часто и довольно убедительно формируется катехизис всего того, что «мы не можем сделать», который в итоге даже способен превратиться в доктрину, вне зависимости от его истинности. Потому проектировщикам взаимодействия следует со здоровой долей скептицизма относиться к подобным предположениям о невозможности исполнения чего-либо. Снова и снова нам удается обойти эти ограничения только лишь потому, что мы не соглашаемся принимать их за чистую монету.

Безусловно, бывают и самые настоящие ограничения, обойти которые не представляется возможным, однако с чередой попыток их преодоления можно приобрести ценный опыт. Даже в тех случаях, когда нам недостает ловкости, чтобы обойти препятствие, путешествие по тупиковому пути может помочь обнаружить какие-то новые возможности, упущенные ранее. Такой подход основан на применении латерального мышления, описанного в работе Эдварда де Боно[31].

Программистов можно назвать королями практичности. Свойственный им прагматизм почти не оставляет шансов для размышлений о невообразимых вещах. Тем не менее эта сильная сторона их личности может оказаться и их слабостью, потому что в некоторых случаях подход, основанный только на практичности, не поможет решить проблему. В процессе изобретения чего-либо инженеры приходят к нужному решению через цепочку практичных шагов и изучение реалистичных вариантов. В результате найденные решения, как правило, являются производными от старых изначальных вариантов, что, как правило, не слишком хорошо.

Мы используем другой подход и просто-напросто допускаем существование любых вариантов, на этом и базируется наше проектирование. Обходя таким способом возможные ограничения, мы можем с большей ясностью взглянуть на персон и их цели, а также придумать решения, которые мы не смогли бы получить при традиционном подходе.



Инженеры чувствуют себя неуютно, когда им приходится сходить с твердой основы рациональности, и оттого предпочитают придерживаться своих ограниченных воззрений. Им *прекрасно известно*, что рано или поздно нам все равно придется столкнуться с этими ограничениями, потому инженеры считают своим долгом отстаивать их. Они называют себя «адвокатами дьявола». Я ценю их заботу, однако ограничения реального мира – это вовсе не та вещь, которую нужно пестовать. Реальный мир не нуждается в таких адвокатах, потому что мы в любом случае не сможем им пренебречь. Последнее слово всегда за реальностью. Осознавая, что решающий ход все равно остается за реальным миром, мы понимаем, что вне зависимости от того, какие варианты мы себе вообразили, они никогда не станут реальностью, если их реализация в принципе невозможна. Но подобные нереализуемые вещи спроектирует лишь человек, у которого отсутствует кровный интерес в игре. Куда больше, мы очень часто обнаруживаем, что все ограничения надуманны и иллюзорны. И мы бы не увидели этого, пока не попытались бы их обойти.

Кейс: ScanMan от компании Logitech

Наш инструмент проектирования под названием «Волшебный компьютер» оказался особенно эффективен в одном глобальном проекте. К нам обратился департамент корпорации *Logitech* из Фримонта, штат Калифорния, занимающийся выпуском сканеров. Нас попросили принять участие в проектировании программного обеспечения для совершенно нового поколения сканеров настольного типа, предназначенных для дома и небольших офисов.

В основе нового сканера *Logitech*, которому мы дали кодовое название «Павлин», лежала технология сканирования нового поколения, а сам он присоединялся к компьютеру через USB-порт. Небольшое устройство размером со свернутую в трубку газету стоило довольно недорого и удобно умещалось на письменном столе. В паз сканера нужно было вставить любой одностраничный документ, после чего этот документ прокручивался через сканер при помощи небольшого моторчика с одновременным сканированием по ходу прокрутки.

Философия компании уже давно была построена вокруг выпуска небольших комплектующих для аппаратного обеспечения, значительную ценность которым придавало программное обеспечение, входящее в комплект. С точки зрения инженеров компании *Logitech*, такая идея выглядела вполне здраво. А вот с точки зрения пользователей, подход не слишком хорош. Он не был целеориентированным.

Специалистам *Logitech* казалось, что дополнительный программный функционал добавляет ценности аппаратному устройству. По крайней мере – продолжали они свою мысль – добавить новые опции в программу обойдется куда дешевле, чем новый функционал в аппаратной части. Такой взгляд на проблему рассматривает соотношение «стоимость/выгода» с позиции производителя продукта, а не со стороны пользователя.

Предшественник нашего «Павлина» был буквально переполнен различными функциями, а у каждого участника команды – маркетолога, менеджера по продукту, программиста или старшего менеджера – непременно были самые любимые функции, которые они яростно защищали на каждом совещании по стратегии. Но если какой-либо продукт и нуждался в принудительном вырезании опций, то «Павлин» был для этого самым подходящим кандидатом.

Мы довольно редко прибегаем к урезанию количества функций в продукте с целью улучшить взаимодействие, однако в отношении «Павлина» убеждение компании *Logitech*, что с новыми опциями продукту добавилась значительная ценность, было заблуждением. С помощью персон и сценариев мы совершенно ясно осознали, что интерфейс продукта был перегружен ненужным, нежеланным и неиспользуемым функционалом.

Мы традиционно начали процесс проектирования с создания набора персон. Расскажу, как мы работали над его созданием.

Розничная цена сканера составляла порядка 150 долларов. Для рынка конечных потребителей это был достаточно производительный сканер с высоким разрешением и качественной цветопередачей, однако поставить его в один ряд с профессиональными планшетными

сканерами, которые в то время стоили порядка 800–1000 долларов[32], было затруднительно. Было абсолютно понятно, что основным рынком для подобного продукта станут пользователи в небольших или домашних офисах, обычно называемых одной аббревиатурой – SOHO (small office, home office).

Малкольм, Воин интернета

Мы создали персону Малкольма – Воина интернета, представителя пользователя из сегмента SOHO. Малкольм – молодой человек, открывший на дому небольшую компанию по разработке сайтов. Он не слишком хорошо разбирается в технических вопросах и не является графическим дизайнером, но неплохо обращается с компьютером и понимает, что оптимизированные картинки, которые быстро загружаются в интернете, гораздо лучше тяжелых полноцветных изображений. С помощью сканера «Павлин» он может получать изображения среднего разрешения и с легкостью встраивать их в веб-сайты, без лишних дополнительных затрат и каких-либо возможных проблем.

Чед Марчетти, мальчик

Сканер «Павлин» также обладал привлекательностью и для тех владельцев домашних компьютеров, кто хотел сканировать изображения больше для личных, нежели для рабочих целей. Для отражения образа домашнего пользователя мы создали персону Чеда Марчетти – мальчика десяти лет, которому нужен был сканер для оформления домашних заданий красивыми цветными иллюстрациями.

Магнум по прозвищу DPI

Мы понимали, что профессиональные графические дизайнеры отдадут предпочтение планшетным сканерам за тысячу долларов, а потому сместили приоритеты с этого сегмента рынка. Однако также мы понимали, что не можем полностью проигнорировать этот сегмент, ведь все великое начинается с малого. У молодого графического дизайнера, занимающегося фрилансом и только начинающего работать на себя, еще не будет средств на покупку серьезного устройства, а потому сканер «Павлин» поможет ему преодолеть первые год-два работы, пока он не сможет позволить себе полноценное профессиональное оборудование. Но это произойдет лишь в том случае, если «Павлин» позволит добиться нужной производительности.

Образ начинающего специалиста с огромным потенциалом мы выразили через персону, которую назвали «Магнум по прозвищу DPI». (Такое имя представляет собой игру слов, основанную на названии старого телевизионного шоу Тома Селлека «Magnum, P.I.»[33] и сокращении от dots-per-inch – распространенной меры плотности точек цифрового изображения.) Возможно, Магнум – представитель не самого крупного сегмента пользователей, но он, несомненно, персона невероятно значимая. Все его друзья, у которых есть домашние компьютеры, всегда обращаются к нему за советом, когда дело доходит до программ для работы с графикой или периферийного оборудования. Через год-другой Магнум сможет обзавестись планшетным сканером, но пока ему будет достаточно «Павлина».

Ни Малкольм, ни Чед особо не разбираются в нюансах обработки графики. Малкольм слишком озадачен другими вещами вроде создания сайтов и зарабатывания денег. У Чеда тоже свои проблемы – ему бы не потерять все отсканированные изображения в недрах файловой системы. Ни тому, ни другому нет дела до настроек разрешения. И тому и другому нужно лишь сканировать изображения, выполнять обрезку под нужные размеры и вставлять в документ. Искомым результатом для них являются именно документы, а не сами изображения. Мы выявили, что обе эти персоны преследуют три важные цели:

- Они не хотят разбираться в разрешениях изображений, сканерах и их настройках.
- Они хотят легко отыскивать на своем компьютере отсканированные изображения.
- Они хотят быстро и просто экспортировать изображения в документы и в другие программы.

Магнум по прозвищу DPI – человек другого толка: он *прекрасно* разбирается в разрешениях и *умеет* обрабатывать изображения. Зная это, было бы логичным предположить, что добавление соответствующего функционала в продукт порадует Магнума. Тем не менее у Магнума уже установлена программа Adobe Photoshop. Это мощная, сложная и очень дорогая программа для обработки изображений, и она является его основным инструментом, который он изучил вдоль и поперек[34]. Он использует Photoshop при каждой возможности, вне зависимости от сложности задачи. Любая попытка «Павлина» предложить схожий функционал и потягаться мощью с Photoshop будет обречена на поражение. Подобно Пи-Ви Герману[35], ступившему на ринг против чемпиона Джорджа Формана, «Павлин» не продержался бы там и одного раунда. Не стоит и пытаться тратить силы на то, что не будет использоваться и станет для нас только лишь обузой.

Как бы то ни было, две из трех целей Магнума совпадают с целями Чеда и Малкольма: он хочет с легкостью отыскивать полученные изображения, и он хочет быстро и просто экспортировать изображения в другую программу (Photoshop).

Единственная настройка, которая может потребоваться Магнуму в процессе сканирования, – это указать физическое разрешение сканера в точках на дюйм. В сканерах предыдущего поколения Магнум мог сэкономить время, сканируя с более низким разрешением, что ускоряло весь процесс. «Павлин» – сканер нового поколения – обеспечивает еще более высокую скорость сканирования, при этом сохраняя максимальное и довольно достойное разрешение в 200 dpi. Получается, что при полноцветном сканировании на один лист формата A4 потребуется приблизительно 20 секунд. Так что, если Магнуму придется потратить 10 секунд на уменьшение разрешения, получив более низкое качество и всего 5 секунд экономии времени сканирования, это не принесет никакой пользы. Разве станет кто-нибудь – даже будь это Магнум – уменьшать разрешение, если скорость сканирования и так высока, а разрешение будет максимальным? Эта мысль явилась для нас озарением и позволила понять, что цели всех трех персон согласуются друг с другом, а потому мы можем сделать счастливыми всех троих и при этом избежать добавления многочисленных функций.

Представляем себе волшебный компьютер

На наших сессиях мозговых штурмов мы проделали упражнение «Волшебный компьютер» и поняли, что Чед будет вполне доволен, если сможет помещать изображения в компьютер даже без сканера. В результате этого упражнения нам стало ясно, что единственная вещь, с которой не хочется разбираться ни Чеду, ни Малкольму, ни Магнуму, – это аппаратные настройки. С этой точки зрения мы легко смогли осознать, что интерес Чеда состоит только лишь в самом отсканированном изображении и только после того, как оно появилось в его компьютере. Его не беспокоит, каким образом изображение попадет в компьютер – хоть по мановению волшебной палочки, но, как только оно там окажется, Чед должен суметь найти его, обрезать и импортировать в другую программу.

Значительная часть устройств конкурентов, равно как и предшественник «Павлина», просто-напросто погружали изображение – а вместе с ним и пользователя – в пучину файловой системы Windows, предоставляя человеку ту же самую иерархию файлов для хранения, обработки и поиска полученных изображений. Однако эту файловую систему на самом деле слишком сложно применять, фактически она бесполезна.

Для работы по правилам этой файловой системы требуется, чтобы Чед задал отсканированному изображению имя, после чего указал директорию, в которой требуется сохранить файл. Если бы ему понадобилось найти это изображение, он должен был бы вспомнить название или место, куда он его сохранил. Реальность такова, что Чед, как и любой другой человек, не слишком силен в запоминании подобных мелочей. Компьютер же, с его жестким диском, приспособлен для таких вещей куда больше, только ему до человека дела нет. Вместо этого он вынуждает Чеда самостоятельно выполнять работу по запоминанию имени файла и места его хранения.

В нашем проекте программа сканера никогда не потребует от Чеда задавать имя файлу и указывать место сохранения. Взамен она молча возьмет картинку и позаботится о ее сохранности вместо Чеда. Когда же ему впоследствии нужно будет отыскать изображение, помимо того, что он легко опознает его по миниатюре, он также сможет найти его по любому из параметров: дате

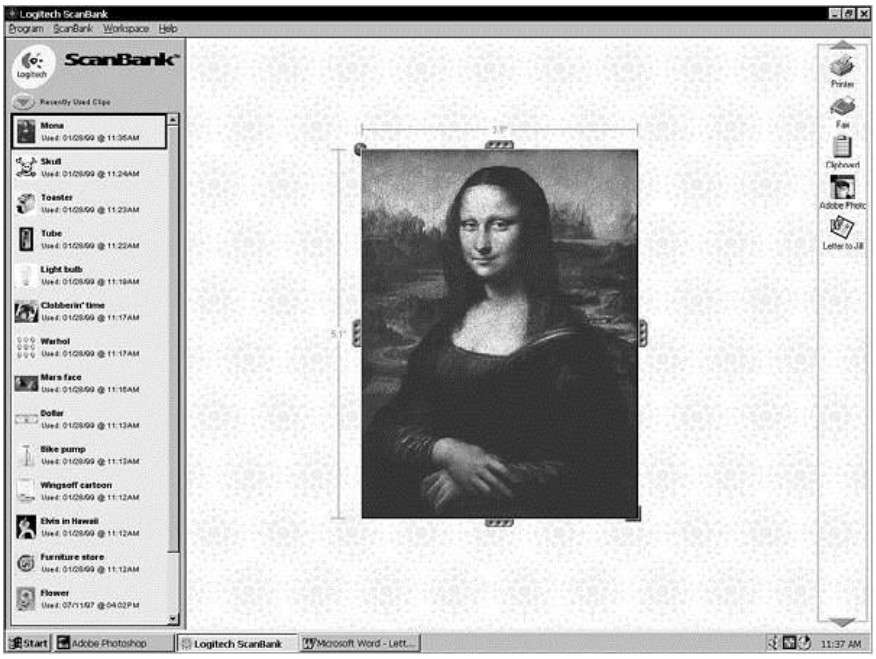
сканирования, размеру, наличию в нем текста или по отметке об экспорте в какую-либо другую программу.

Вместо того чтобы заставлять Чеда и Магнума разбираться в различных настройках сканера, мы сфокусировались на трех более важных моментах:

• убрали из интерфейса все управляющие элементы;

• сделали невозможным потерять изображение в пучинах файловой системы;

• максимально упростили экспорт полученных изображений в документы или другие программы.



Мы провели ревизию всех доступных опций для работы с изображениями и пришли к выводу, что из них жизненно необходимы лишь три – от всех прочих можно смело отказаться или схожие манипуляции можно произвести с изображением позже в других, более подходящих программах наподобие Photoshop.

Вот опции, которые мы сохранили:

• обрезка (Crop): усечение краев изображения;

• масштабирование (Resize): изменение размера изображения;

• ориентация (Reorient): вращение изображения, изменение его ориентации.

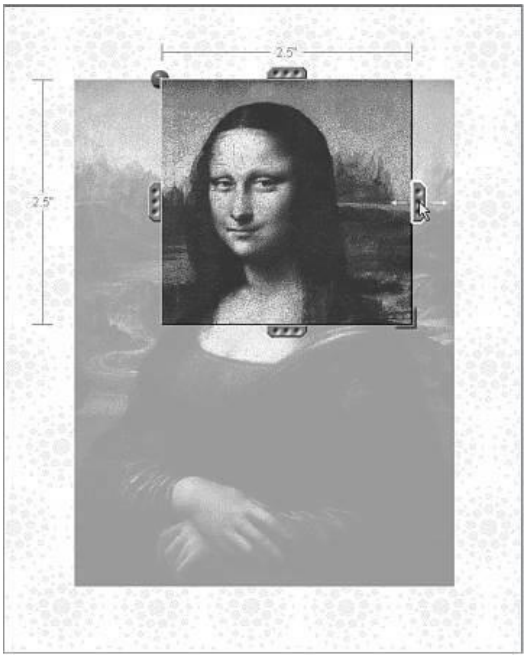
Набор опций получился достаточно небольшим, однако это были самые необходимые опции, которые бы использовались чаще всего, поэтому мы решили сделать их качество наивысшим и обеспечить простоту применения. Существенная экономия на объеме кода за счет отсутствия необходимости программировать множество функций позволила команде разработчиков освободить время и вложить максимум усилий в три оставшиеся опции.

Первоклассная обрезка изображения

Все инструменты программной обрезки изображений, которые мне довелось испытать, работают одинаково некорректно. Пользователь нажимает левую кнопку мыши на картинке и, удерживая кнопку, растягивает прямоугольную рамку. Первая точка, где происходит нажатие кнопки мыши, оказывается левым верхним углом нового изображения, а точка, где пользователь отпускает кнопку, завершая операцию растяжения рамки, оказывается правым нижним углом изображения. Таким образом, все, что остается за пределами этой рамки, обрезается, а

выделенная область становится новым изображением. Этот метод является простым, быстрым, легким в реализации и несложным в понимании. К примеру, в серьезной программе обработки графики Photoshop тоже используется этот метод. Как бы то ни было, у этого способа есть ряд больших недостатков. Самый большой из них заключается в том, что выполнить операцию выделения области весьма непросто: требуется произвести манипуляцию с величайшей точностью всего за одно движение мыши. При этом, если три стороны изображения пользователь сможет обозначить довольно легко, то вот выделить четвертую сторону, не нарушая уже установленных границ трех остальных, может оказаться затруднительно. Более того, так как данная операция необратима, получается, что программа не позволит получить еще один вариант обрезки на основе того же изображения.

Предложенный нами инструмент обрезки изображения позволил решить обе упомянутые проблемы с помощью простого, легкого в освоении и несложного в понимании способа. Каждая из четырех сторон изображения содержала собственный маркер, видимый во время всей операции обрезки. По маркеру можно было ясно понять, что это объект непосредственного манипулирования. Все, что теперь требовалось сделать Чеду, – это потянуть за маркер, после чего он мог моментально получить визуальную обратную связь на свои действия и оценить последствия. С перемещением маркера та часть изображения, которая остается позади него, окрашивается в призрачно-серый цвет. Становится ясно, что изображение обрезается, но также остается понимание, что эта операция не является необратимой. Чед может с той же легкостью вернуть маркер обратно и таким образом восстановить соответствующий фрагмент изображения в полном цвете.



Перетягивая один из маркеров, Чеду сразу становится понятно, что все четыре границы области обрезки не зависят друг от друга, то есть смещение одной границы не влияет на остальные. Он может сколько угодно раз перемещать границы области обрезки до получения нужного результата. Ощущение от такого пользовательского взаимодействия совсем иное, нежели при использовании традиционных инструментов обрезки изображения, в которых операция удаления краев слишком зависит от обстоятельств, ее нельзя обратить и она должна быть выполнена максимально точно за одно движение. Пользователей, обладающих непревзойденной ловкостью рук, способных совершить такое точное движение, весьма немного. Десятилетний мальчик Чед явно не относится к этой категории. Кроме того, наш вариант обрезки обладает наглядностью и позволяет совершать операцию в несколько заходов. Ведь даже самые лучшие дизайнеры обычно предпринимают несколько попыток, прежде чем получат идеальный

результат. С прежними инструментами сделать такое было просто невозможно. Созданный же нами инструмент для *Logitech* справлялся с этим превосходно.

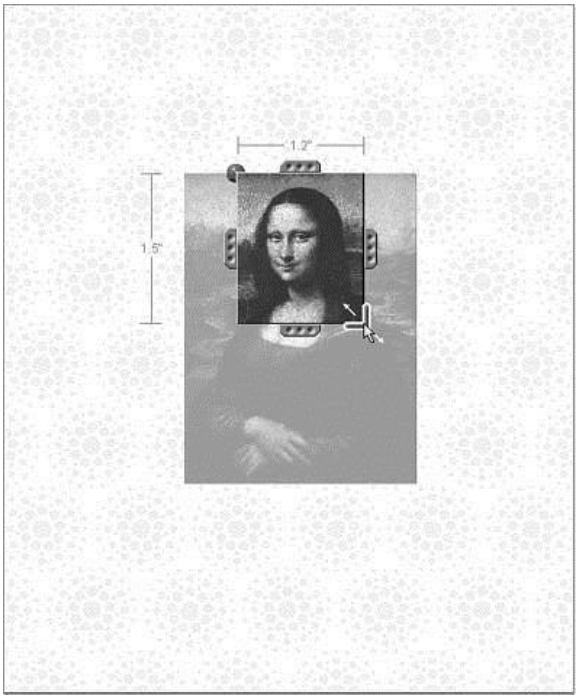
Даже после того, как Чед подтверждал обрезку выбранного фрагмента, операция не становилась необратимой. Текущие параметры обрезки просто становились атрибутами изображения, а само оно постоянно хранилось в первоначальном варианте. (В меню также присутствовал дополнительный пункт, который позволял сделать обрезку необратимой в целях экономии дискового пространства.) Таким образом, Чед мог создать скан-копию фотографии всех членов семьи, а затем вырезать из нее только изображение матери, если того требовало домашнее задание, а спустя три месяца снова найти то же отсканированное фото и изменить границы обрезки, сохранив только изображение отца для вставки в текст письма. Воспользуйся Чед любой другой программой для получения сканов изображений, ему пришлось бы сканировать фотографию заново.

Первоклассное масштабирование изображения

Для того чтобы изменить размер изображения, большинство программ для обработки графики попросит указать новые размеры в диалоговом окне. Такой подход призван обеспечить высокую точность и позволяет изменять пропорции изображения. Тем не менее точность редко на самом деле важна, а непропорциональное изменение изображения и вовсе нежелательно. Диалоговое окно предлагает невостребованные опции вместо тех, что действительно *нужны*, например возможность увидеть, каков будет новый размер изображения относительно прежнего. Инструмент, отвечающий за масштабирование, должен обеспечивать наглядность.

Предложенное нами решение для масштабирования изображения представляло собой небольшой уголок красного цвета, размещенный в правом нижнем углу полученного со сканера изображения. Если навести на этот уголок курсор мыши, он слегка увеличивается в размерах, всего на пару пикселей. Такое поведение я называю «гибким откликом» – по нему Малкольм может понять, что с этим объектом можно производить непосредственное манипулирование. Стоит Малкольму нажать кнопку мыши и потянуть за уголок, изображение прямо в режиме реального времени изменит свои размеры, увеличиваясь или уменьшаясь в зависимости о направления движения. При этом пропорции изображения всегда будут сохраняться. Изменение пропорций – это уже задача Магнума, а для этих целей он обычно применяет программу Photoshop.

Еще больше пользы добавляют этому инструменту указатели размеров, которые появляются вдоль сторон изображения. Значения на этих указателях изменяются сразу же, как только Малкольм тянет за уголок, тем самым обеспечивая моментальную обратную связь, информирующую о текущих размерах изображения. Через дополнительный пункт в меню Малкольм может выбрать другие единицы измерения вместо дюймов – пиксели или метрическую систему.



Первоклассный поворот изображения

В типичных программах для работы с графикой можно также увидеть опцию поворота изображения.

В основном эта опция применяется в трех случаях:

• для изменения общей композиции изображения путем вращения отдельных его фрагментов;

• для корректировки выравнивания изображения, отсканированного с отклонением от вертикали;

• для восстановления перевернутого изображения или изменения его ориентации.

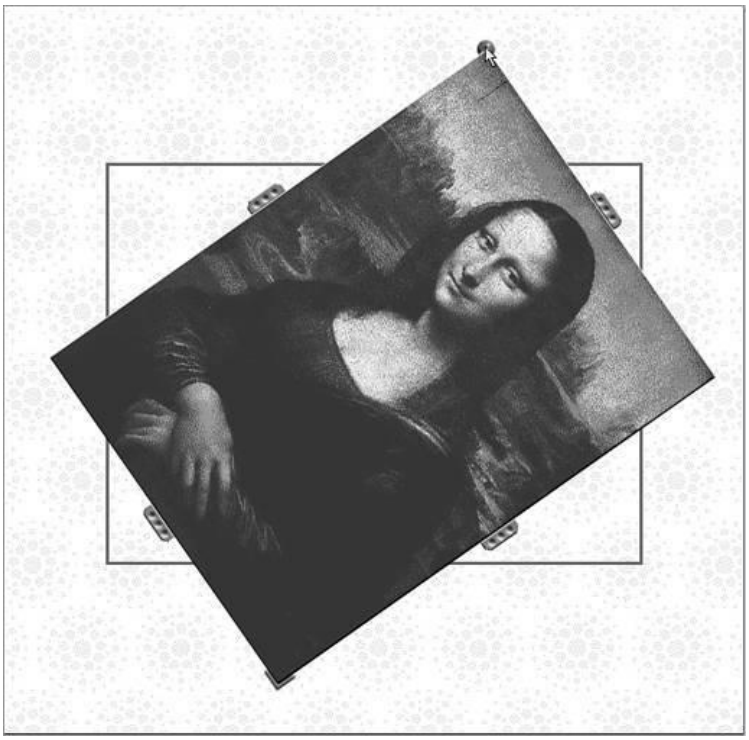
В функционал большинства сканирующих программ, в том числе и у тех, что поставлялись с предшественниками «Павлина», входил инструмент поворота, способный решить все три упомянутые задачи. Мы оценили всю мощь и сложность этих программ взглядом Чеда, Малкольма и Магнума и решили выбрать кардинально иной подход.

От первой задачи из списка мы отказались сразу. Подобное применение инструмента пригодилось бы только дизайнеру, а такого в числе наших пользователей не было. Больше всего под описание дизайнера подходил Магнум, однако он применил бы для этих целей мощный инструментарий программы Photoshop.

Вторая опция из списка, называемая *выравниванием*, не работала бы хорошо ввиду ограничений самой технологии. Практически все устройства на основе растровой графики – дисплеи, сканеры и принтеры – отрисовывают прямые линии, всего на один или два градуса отклоненные от вертикали или горизонтали, в виде искаженных зазубренных линий, такой артефакт носит название «алиасинга» (ступенчатости) или «зубчатости». Подобные линии не подвергаются распрямлению с помощью компьютерной обработки, а применение стандартной опции поворота приводит к возникновению умопомрачительных оптических коллизий. Как говорится, лекарство бывает хуже самой болезни. Но это еще не самое плохое. Самое плохое – то, что описать поворот изображения на пару градусов в программном коде можно, только если применить невероятно сложные и изощренные методы. Значительная часть других программ для сканирования с большой гордостью преподносит эту совершенно бесполезную во всех отношениях опцию, и это прекрасный пример того, что По Бронсон называет «чем меньше они видят, тем лучше это для них».

Если скан-копия изображения получается с отклонением на один или два градуса – гораздо проще и быстрее устранить этот огрех путем выравнивания листа при повторном сканировании. Внутреннее устройство сканера «Павлин» не только облегчает пользователю эту задачу за счет точно выровненных роликов и высокой скорости сканирования, но и делает практически невозможным изначально вставить лист в паз сканера неровно.

Третий случай использования опции поворота – это изменение ориентации изображения. Пользователь может довольно легко ошибиться и отсканировать изображение «лежащим» на боку или вовсе вверх ногами. Программа позволяет просто и быстро выполнить вращение изображения на 90, 180 или 270 градусов, приведя ориентацию к правильному варианту.



Потому мы дали инструменту имя «Ориентация» вместо «Поворот» и снова приложили максимум усилий, чтобы обеспечить первоклассное качество. В левом верхнем углу изображения мы поместили небольшой синий круг, схожий с красным уголком для масштабирования изображения. Стоило пользователю навести мышь на этот синий круг, он слегка увеличивался, и это снова сигнализировало о том, что с ним можно производить манипуляции.

Если Малкольм нажмет кнопку мыши на этом синем круге и потянет за него, то вокруг изображения возникнет ярко-зеленая рамка. Она называется «прицел» и показывает, как будет выглядеть изображение после изменения ориентации. Стоит Малкольму перетянуть изображение чуть дальше, чем первый угол ярко-зеленой рамки, как прицел изменяется, показывая следующее возможное положение для выравнивания – 90, 180 или 270 градусов. Так Малкольм может заранее предугадать, какой результат получится после выполнения операции. Ему тут же становится совершенно ясно, что изображение поворачивается только в определенные позиции, а свободное вращение, равно как и выравнивание искаженного изображения, невозможно. Все персоны моментально понимают, как работает этот инструмент.

Первоклассный результат

Пользовательское тестирование продукта, проведенное нами по просьбе клиента, выявило одну весьма примечательную вещь. Согласно нашим ожиданиям, всем участникам тестирования новый интерфейс должен был показаться невероятно простым в использовании и понимании, а

потому привлекательным. Но поразительным было то, что все участники единогласно высказались о «Павлине» как о «самом мощном» продукте. Учитывая его скромное количество опций, это суждение нельзя назвать справедливым. Но вот исходя из эффективности этих опций, которая явно ощущалась пользователями, продукт действительно стал более мощным.

С выходом продукта «Павлин» (настоящее название – ScanMan) на рынок, специалистов отдела технической поддержки *Logitech* охватило крайнее беспокойство, поскольку звонков от пользователей, недоумевающих, как пользоваться продуктом, было значительно меньше, чем обычно при выходе новых продуктов.

Разрыв между устройствами и программами

Со стороны проектировщика пользовательского взаимодействия разделение между аппаратными устройствами и их программами не имеет смысла, поскольку для пользователя они также неделимы. Пользователю не важно, что из этого обходится дороже в производстве. Поэтому проектировщики взаимодействия могут эффективно решить проблемы, возникающие в процессе разработки гибридных продуктов.

Среди инженеров-разработчиков можно выделить две категории специалистов: первые являются разработчиками аппаратной части устройств и создают электронные платы и микрочипы, а вторые – программными разработчиками и пишут программный код. И хотя плоды их деятельности обретают единство в составных – гибридных – продуктах, обе эти отдельные фракции разработчиков, как правило, не взаимодействуют между собой. Иногда они даже не коммуницируют друг с другом, а просто «перекидывают» готовые модули через воздвигнутый между ними «забор».

Так исторически сложилось, что инженеров по аппаратной части в большинстве компаний, специализирующихся на производстве гибридных продуктов, в разы больше, чем разработчиков ПО. Однако с увеличением доли аппаратных устройств в мире инженеры и сами устройства начинают утрачивать свои позиции. В противовес им все большую ценность в глазах пользователей обретают уникальные возможности программного обеспечения. Такая ситуация порождает вынужденное перемирие в большинстве компаний – производителей гибридных продуктов.

Hewlett-Packard – хороший пример компании, где преобладают разработчики аппаратной части устройств. Принтеры, которые производит компания, – это великолепный продукт, чудо инженерной мысли, однако спустя два десятилетия непрерывных улучшений ни одно из этих устройств не способно в полной мере наладить взаимодействие с моим компьютером. Они не могут сообщить компьютеру о количестве бумаги, оставшейся в лотке, или количестве порошка в картридже, равно как и о количестве заданий в очереди на печать. Такая пренебрежительная невнимательность к потребности человека обладать информацией – это улика, по которой можно легко выявить компанию, где преобладают разработчики устройств.

По иронии судьбы у компаний, занимающихся разработкой аппаратных устройств, гораздо больше опыта в привлечении сторонних специалистов из области промышленного дизайна с целью сделать продукт более полезным и желанным для пользователя. Разработчики же программного обеспечения склонны предпринимать попытки справиться с проблемой самостоятельно. Любая компания – производитель гибридных продуктов, в которой нет проектировщиков, способных выступить посредниками между обеими сторонами, в результате выпустит продукт, не отвечающий потребностям пользователей. Такая ситуация абсолютно ясно видна в большинстве примеров из главы 1 «Загадки информационной эры».

С увеличением количества продуктов, построенных по гибридной модели, увеличивается и потребность в целеориентированном проектировании, поскольку оно не зависит от возможных способов реализации продукта.

Корпорация *3Com* – разработчик продукта PalmPilot – хороший пример компании с гибридным подходом, которой удалось оптимальным образом объединить разработку программной и аппаратной части посредством проектирования. Одно касание экрана – и устройство «просыпается» ровно в том же состоянии, в каком оно пребывало, когда выключилось. Когда устройство моментально откликается на действия пользователя, это явный признак того, что проектирование аппаратной части согласуется с потребностями программного обеспечения. Есть и обратный пример: моей фотокамере Nikon CoolPix 900 требуется долгих

семь секунд для загрузки после каждого включения, и это при отсутствии жесткого диска. Когда устройство настолько инертно, становится очевидным, что бал правят разработчики аппаратной части.

Конечно, в мире реального проектирования ситуация такова, что большинство компаний – разработчиков программ по вполне понятным причинам держатся в стороне от аппаратного обеспечения. Проектировщики такое решение поддерживают даже в тех случаях, когда специализированное оборудование только выиграло бы от потенциального сотрудничества.

Тем не менее при достаточном бюджете проекта проектировщикам не стоит отказываться давать советы относительно аппаратной части продукта. P@ssport – система развлечений в полете от корпорации *Sony*, описанная в главе 9 «Проектируем для удовольствия», функционировала на основе специализированных компьютеров, так что поставщик имел полную власть как над аппаратной, так и над программной частью всей системы. Проектировщики из моей команды дали корпорации *Sony* несколько рекомендаций по аппаратной части этого проекта.

Продукт Drumbeat от компании *Elemental*, процесс проектирования которого я описывал в главе 10 «Проектируем для результата», предназначался для запуска на стандартных настольных компьютерах на базе Wintel, так что в этом случае мои проектировщики даже не пытались давать рекомендации.

В нескольких клиентских проектах, к которым, в частности, относится и «Павлин» от *Logitech*, моей проектировочной команде посчастливилось принять участие в повышении ценности аппаратной части устройств. Каждая компания обладала возможностью совершить рискованный шаг в сторону гибридных продуктов, принимая все потенциальные опасности и вероятные выгоды, сопутствующие этому решению.

Меньше значит больше

Программисты, безумно увлеченные всяческими гаджетами и склонностью все контролировать, обожают начинять продукты многочисленными «примочками» и опциями, однако подобное стремление идет вразрез с фундаментальными представлениями о качественном проектировании. Как говорится, меньше значит больше.

Чем меньше подозревает пользователь об участии проектировщика в разработке продукта, тем больше это сигнализирует о том, что проектировщик справился со своей задачей на высочайшем уровне. Качественный интерфейс, как и обслуживание в ресторане высшего класса, должен быть ненавязчивым. Если какой-то аспект взаимодействия удался проектировщику особенно хорошо, пользователи даже не должны обратить на это внимания. В индустрии программного обеспечения, где целью проектирования провозглашаются «крутые» интерфейсы, бывает весьма трудно продираться сквозь тьму артефактов взаимодействия, на которые какой-нибудь несчастный программист, очевидно, потратил целую уйму времени. Как жаль, что он не потратил свои усилия на создание чего-либо более эффективного. Многим графическим дизайнерам свойственно полагать, что качественное проектирование означает «крутой дизайн». Отчасти это действительно бывает так, однако *вне зависимости от того, насколько «крут» ваш интерфейс, чем он меньше заметен, тем лучше*[36]. Еще раз: суть подхода заключается в том, что чем менее навязчив интерфейс для пользователя, тем качественнее справился проектировщик со своей работой. Представьте, что вы смотрите фильм, на границах кадра заметны яркие софиты, а после каждой сцены слышно, как режиссер кричит «Снято!». Только представьте, как навязчиво будут выглядеть подобные «спецэффекты» и как все волшебство кинофильма для зрителя враз будет разрушено.

Величайший программист и проектировщик программного обеспечения Кай Краузе (Kai Krause) известен своими неповторимыми интерфейсами. Кай разработал ряд программ для работы с графикой, которые считаются одними из самых мощных и интересных продуктов. Интерфейсам его продуктов свойственна необычайная ошеломительная красота и одновременно некая загадка, будто играешь в игру. В дополнение к своей специализации программиста Кай является также графическим дизайнером, поэтому его продукты отражают дизайнерское стремление добавлять своим творениям элемент неясности – словно в современном искусстве – с целью произвести впечатление. В отношении Кая такой подход работает «на ура», поскольку пользователями его продуктов являются такие же дизайнеры – как профессионалы, так и люди,

увлеченные дизайном в качестве хобби. А вот за пределами этого мира продуктам Кая найти понимание нелегко.

* * *

Программирование характеризуется тем, что для решения любой поставленной задачи в нем можно найти бесконечное количество способов. Опытные программисты в своих поисках оптимального варианта решения периодически наталкиваются на такой вариант, который позволяет избавиться от сотен и даже тысяч строк кода. Но это происходит лишь тогда, когда программист способен мысленно совершить качественный скачок и заглянуть вперед. Стоит ему избавиться от множества строк кода, как его программа станет на порядок лучше. Чем меньше кода, тем меньше ошибок, запутанных моментов, возможностей для возникновения некорректного поведения, и такую программу легче сопровождать.

Проектировщики пользовательского взаимодействия такой подход разделяют. В поисках оптимальных вариантов проектирования они выявляют участки взаимодействия, в которых можно избавиться от целых экранов или больших и сложных диалогов. Проектировщику хорошо известно, что каждый дополнительный элемент интерфейса – дополнительная нагрузка на пользователя. Каждая лишняя кнопка или значок – это еще один элемент, с которым пользователь должен ознакомиться и попытаться освоить, чтобы достичь своих истинных целей. Добиться большего эффекта при меньших усилиях – это всегда более выигрышный вариант.

Проектировщик, который мыслит в правильном направлении, будто бы вовсе *исключает* интерфейс из продукта. Он не станет проектировать экран за экраном, испещренный многочисленными кнопками и «примочками». Как-то к нам обратился менеджер по продукту одной из крупных компаний, допытываясь, можем ли мы перепроектировать его продукт. Далее мы выяснили, что в интерфейсе должно быть около десятка диалоговых окон. Тогда мы рассказали ему, как происходит наш рабочий процесс, а затем огласили предварительную стоимость работ. Если я правильно помню, цифра составляла приблизительно 60 000 долларов. Услышав это, менеджер пришел в ярость. «Возмутительно! – воскликнул он. – Пять тысяч долларов за один экран!» У меня тогда просто духу не хватило сказать, что когда мы урежем количество диалоговых окон, оставив всего одно или два, то цена даже возрастет, если использовать для ее вычисления подобный метод. Он бы просто не понял, о чем речь. Рассчитывать стоимость проектирования, исходя из стоимости каждого экрана, – это то же, что оплачивать счет в ресторане, прибавляя цену за каждый подход официанта к столику. Чем лучше работает официант, тем *меньше* он перемещается по залу, а чем лучше работает проектировщик, тем «меньше» интерфейса он создает.

Бывают в жизни проектировщиков взаимодействия и в некотором роде обидные моменты. Например, если вы в рабочем процессе нашли, как можно сделать что-то невероятно правильно, обстоятельно и до невозможности верно, то все, завидев это, восклицают: «Ну конечно! А разве это можно было сделать как-то *по-другому*?» Такое случается даже в тех ситуациях, когда клиент многие месяцы или годы безуспешно пытается найти решение проблемы, бездумно глядя на нее без малейшей идеи в голове. Такое бывает даже тогда, когда наше решение приносит компании миллионы долларов дохода.

Реализацию значительной части по-настоящему прорывных инновационных идей *поначалу очень сложно даже вообразить, зато стоит ей появиться на свет, как всем сразу становится очевидно, что это нужно было делать именно так*. Совершить такой прорыв в проектировании невероятно сложно. Можно потратить долгие часы на подготовку, обучение и исследование проблемной области и все равно не найти подходящего решения. А потом вдруг кто-то другой укажет на ключевой момент, и все сразу встает на свои места, столь же естественным образом, как и с изобретением колеса. Вы можете бить во все колокола, возвещая о своей находке, но найдутся те, кто ответит: «Конечно же, колесо круглое! А разве оно может быть каким-то другим?» Так что, как ни обидно это признавать, но похвастать оригинальной находкой в отношении проектирования бывает весьма затруднительно.

Алан Карп, ученый в области компьютерных наук, как-то поведал: «Почти все заявки на патент, которые я направлял, были отвергнуты по причине „очевидности“».

Когда я говорю, что интерфейс должен быть ненавязчивым, я не имею в виду, что в нем должно быть мало функционала, хотя иногда справедлив и такой вариант. Я имею в виду, что не следует вынуждать пользователя взаимодействовать с программой много дольше, чем это необходимо для решения его конкретных задач.

* * *

В этой и двух предыдущих главах я кратко описал наши самые широко используемые инструменты проектирования. Они доказали свою эффективность в проектировании продуктов и сервисов, начиная от систем промышленного контроля до корпоративного планирования и продуктов для массового рынка. В следующей главе я расскажу о некоторых других доступных проектировщику инструментах, призванных помочь создавать продукты с более качественным пользовательским взаимодействием.

Часть V. Вернуть себе бразды правления

12. В отчаянном поиске юзабилити

Со стремительным массовым распространением на рынке устройств, функционирующих на основе программного обеспечения – как компьютерных, так и специализированных, – аудитория пользователей претерпела изменения. Ранее сообщество пользователей было небольшим и состояло из ярых поклонников всяческих технологий. На сегодняшний день это сообщество значительно выросло в размерах – теперь это люди, не слишком хорошо разбирающиеся в технике, нетерпеливые и несчастные. Вероятно, каждому, кто так или иначе связан с разработкой программного обеспечения (да и любому другому человеку), доводилось слышать раздраженную брань пользователей и возникающее в связи с этим желание оказать им *хоть какую-то помощь*. И многие специалисты действительно ринулись на подмогу с полной решимостью исправить эту ситуацию. Все они обладали отличной подготовкой, солидной репутацией и блестящими списками клиентов высшего класса. Но результат их совместной работы оказался пустышкой – их программным продуктам не хватает самой малости: они так и не стали желанными для пользователя. Вследствие этого образовалось крайнее непонимание, какие способы выбрать, чтобы в действительности решить проблему пользователей. В данной главе я постараюсь распутать этот клубок и продемонстрировать пользу различных специализаций, а также показать взаимосвязь с целеориентированным подходом к проектированию взаимодействия.

Координация действий

Последовательность действий в процессе разработки программного обеспечения – это, вероятно, наиболее важный аспект проектирования. С самых первых времен появления процесса разработки ПО последовательность действий представляла собой следующее: программирование, отладка, доработка. Первым делом программист пишет код, а затем выполняет пошаговую трассировку, чтобы выявить непредвиденные ошибки, возникшие в процессе создания программы. После этого он вносит исправления, то есть доводит программу до готовности к развертыванию.

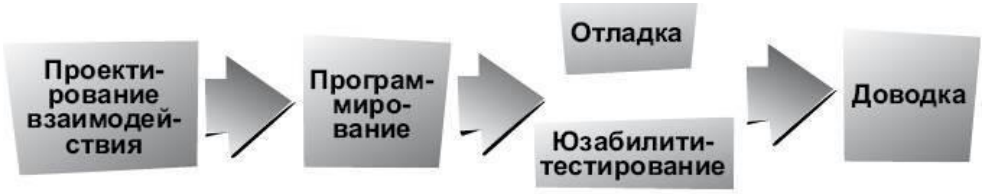
Вот так схема не работает

Вполне нормально, что разработчики гораздо более охотно готовы принять какой-либо новый для них вид деятельности, только если он не нарушает сложившийся уклад. Есть метод под названием «юзабилити-тестирование», он получил весьма широкое распространение, а суть его заключается в том, чтобы проводить исследование с помощью эмпирического подхода, при

котором пользователи осуществляют реальное взаимодействие с продуктом. Этот метод особенно хорошо прижился в индустрии высоких технологий благодаря тому, что с легкостью встраивался в имеющуюся последовательность действий. Такое юзабилити-тестирование возможно выполнить только при наличии уже функционирующей программы, а потому, очевидно, необходимо дожидаться готовности программы к запуску. В этом случае юзабилити-тестирование возможно проводить одновременно с этапом отладки, что представляется удобным. Программисты считают такой дополнительный способ тестирования приемлемым, поскольку он не вносит разлада в имеющуюся последовательность действий.

И так схема тоже не работает

Как я уже упоминал ранее, программирование соотносится с проектированием так же, как заливка бетоном со строительством деревянного каркаса фундамента. Вне зависимости от того, насколько профессионален проектировщик и какие методы он использует, если процесс написания кода уже запущен, большого толку от последующего проектирования практически не будет. Важнейшим постулатом при внедрении проектирования в общий процесс разработки является необходимость проводить этап проектирования до этапа программирования. Очевидно, что я являюсь сторонником целеориентированного подхода к проектированию, тем не менее, если программированию будет предшествовать какой бы то ни было системный процесс проектирования, это будет в разы эффективнее, чем если бы он следовал после этапа программирования.



А это рабочая схема

Если проектирование взаимодействия идет перед этапом программирования, в процессе разработки программного обеспечения возникают кардинальные перемены. Естественно, что эти перемены прямо отражаются на программистах, вызывая у них крайние опасения. Если раньше они были первым звеном в цепочке создания программы, а следовательно, и самым важным, то теперь, раз на первом месте возник другой вид деятельности – не будет ли это означать, что самая важная роль в процессе отдана другим специалистам? На самом деле им нечего опасаться, и мы рассмотрим это подробнее в следующей главе.

За всю свою жизнь я столько раз программировал, изобретал, тестировал, документировал, проектировал, продавал, выводил на рынок и поддерживал программные продукты, что могу с уверенностью утверждать: этап программирования во всей этой цепи – самый сложный и предъявляет невероятно высокие требования к тем, кто принимает в нем непосредственное

участие. (Здесь я имею в виду именно *профессиональное* программирование – то есть разработку программ для коммерческой реализации. Существует известная истина, что сложность программы увеличивается в зависимости от объема кода по экспоненте. В студенческие годы каждый так или иначе пишет небольшие программы, объемом в сто-двести строк кода. И даже многие пользователи создают для решения своих рабочих задач программы приблизительно такого же объема. Но объемы программ, предназначенных для коммерческой реализации, с легкостью могут достигать более чем пятидесяти тысяч строк, потому обычному человеку затруднительно даже представить, какой большой сложностью обладают такие приложения.) Так что, хотя другие специалисты могут оставаться в неведении, для программистов предельно ясно, что именно их работа вносит значительный вклад в общее дело, в разы превосходя вливания других специалистов.

Еще одной причиной, благодаря которой в компьютерной индустрии так широко принята именно такая последовательность, как «программирование – тестирование – доводка», является миф о непредсказуемом рынке, упомянутый в главе 3 «Пустые растраты». Если нам заранее не может быть известно, какие потребности возникнут у рынка, для чего в таком случае впустую тратить лишнее время на проектирование взаимодействия? Пишем программу, выпускаем ее на рынок, а дальше смотрим по ситуации. Более того, при таком подходе мы снимаем с себя всякую ответственность за возможный провал.

Тем не менее даже при наличии всех указанных проблем жизненно необходимо, чтобы более рассудительные люди внесли изменения в имеющуюся цепочку действий посредством внедрения этапа проектирования перед этапом программирования.

Юзабилити-тестирование

В сравнении с творческим процессом, любая другая деятельность должна играть второстепенную роль, в особенности та, что связана с наблюдением. Получается, что программисты занимаются творчеством, а юзабилити безмолвно ставит их у руля со словами: «Сначала вы создадите нечто, а затем я проведу тестирование и смогу оценить, насколько хорошо вы выполнили свою задачу». Однако в связи с тем, что мир высоких технологий вращается с невероятной скоростью, только что созданные продукты моментально выходят на рынок. И тестирование, выполняемое постфактум, уже не сможет что-либо изменить.

Я мог бы сравнить методы юзабилити-тестирования с наждачной бумагой. Сотвори вы стул, наждачная бумага сгладит все его шероховатости. Сделали стол – и вновь наждачная бумага все отшлифует. Однако никакая шлифовка не сделает стул из стола. И все равно мне неоднократно приходится наблюдать, как тысячи людей усердно шлифуют столы посредством приемов юзабилити в надежде получить стулья.

Юзабилити-тестирование до этапа программирования

Бесспорно, вполне возможно проводить этап юзабилити-тестирования до начала этапа программирования, но в этом случае сама сущность и ценность всего процесса будет подвергнута радикальным преобразованиям. При таком подходе тестирование больше похоже на классическое исследование, характерное для университетской среды. Один мой коллега из крупной компании, занятый в сфере разработки ПО, как-то провел типичный юзабилити-тест, в ходе которого ему удалось выявить как сильные, так и слабые места такого предваряющего этапа тестирования. В своем тестировании он намеревался понять, насколько эффективна строка состояния в самом низу окна программы. В ходе эксперимента участникам нужно было выполнить несложное задание в электронной таблице. При этом каждые пять минут в строке состояния отображалась надпись: «К обратной стороне вашего стула приклеена банкнота в $50. Заберите ее!» Тестирование продолжалось целый день, однако за это время не нашлось ни одного участника (а их было более десяти), кто предпринял бы попытку забрать купюру.

Бывает весьма полезно осознать тот факт, что пользователям обычно нет никакого дела до надписей в строке состояния, хотя в программистской среде она столь популярна. Тем не менее это не делает более ясным суть проблемы: что должно представлять собой это состояние, чтобы для пользователя это было действительно полезно? И нужно ли вообще что-то отображать? Где

именно это лучше сделать? На все эти вопросы касательно проектирования ответов по-прежнему нет.

Интеграция юзабилити-тестирования в процесс разработки

В профессиональной литературе приведено множество различных методик и советов относительно проведения тестирования, только они содержат весьма незначительное количество упоминаний способов тестирования на том этапе, когда продукта еще фактически нет. На практике существует необходимость в создании и тестировании хоть какой-то программы. Как правило, в роли макета выступает наскоро сверстанный прототип или некое подобие «кукольного театра» – форма из бумажных вырезок и иных низкобюджетных материалов.

Разумеется, отслеживая реакцию пользователя при демонстрации спектакля в этом «кукольном театре», вы можете узнать много интересного, однако, пропуская этап проектирования, вы в итоге можете прийти к осознанию, что тестировали совсем не то, что нужно. Более того, при такой форме эксперимента любое незначительное действие, исходящее от инициатора тестирования, вроде кивка головой, мимолетно сказанного слова, неосторожного взгляда, может оказать сильное влияние на результаты, искажая их.

Наиболее адекватные результаты можно получить, если прибегнуть к дорогостоящему методу сравнительного тестирования, когда создается два варианта программы и каждая предлагается пользователям. Но даже при таком подходе все, что вы сумеете узнать, – это какой из двух вариантов лучше. Вы по-прежнему не сможете понять, какой в реальности могла бы быть самая подходящая для пользователя программа.

Грамотное юзабилити-тестирование помогает выявить те предположения, которые оказались ошибочны, на взгляд проектировщика. Вариант, при котором вы сначала показываете результаты проектирования пользователям, а затем вносите изменения методом итераций, всегда является более выигрышным, чем полное отсутствие проектирования. Ввиду того что часть новых технологий, например распознавание речи, изучена еще недостаточно хорошо, даже простейшие юзабилити-тесты могут сыграть значительную роль.

Пожалуй, одним из самых ценных моментов тестирования можно назвать присутствие на эксперименте программистов, наблюдающих за битвой пользователей с их разработками из-за одностороннего зеркала. Программистов обычно крайне изумляет поведение пользователей, вплоть до недоверия, и они выражают недовольство: «В тесте участвуют какие-то недоумки!» Так юзабилити-тестирование метко цепляет упрямых разработчиков ПО, указывая им на то, что проблема действительно имеет место. Точно так же оно может открыть глаза и руководителям.

Говоря словами одного популярного рекламного слогана о зубной пасте: с помощью юзабилити-тестирования можно эффективно предотвратить «кариес», если внимательно следовать советам специалистов и придерживаться целеориентированного подхода к проектированию. Главное здесь – не забывать, что все еще остаются факторы, способные оказать более сильное влияние.

Многопрофильные команды разработки

Из-за того что разработчики активно сопротивлялись всему, что, по их мнению, было способно внести разлад в привычную им последовательность действий процесса разработки, в сообществе проектировщиков начали возникать различные хитроумные предложения. Так, широко обсуждается идея того, чтобы в команды, осуществляющие проектирование, включались специалисты из разных предметных областей.

По этой гипотезе, команда, в которую входят представители со стороны пользователей, программистов, руководителей, маркетологов, юзабилити-специалистов, может сработать более эффективно. Исходя из моего опыта, такой метод «круглого стола» не дает сколько-нибудь значимых результатов. У каждого участника собственные цели и проблемы, все они различны, а тот, чьи цели наиболее весомы, обычно не способен понятно излагать свои проблемы. Много хуже – программисты, неизбежно держащие в своих руках тотальный контроль над программными артефактами, в итоге тайно управляют всей командой, словно серые кардиналы.

С помощью круглого стола добиться желаемых перемен невозможно. При всей демократичности, полидисциплинарности и многокультурности такого подхода, когда за бортом

не остается никого, этот метод не может исправить ущербный процесс разработки, при котором пользователи продолжают получать неприятные во взаимодействии продукты.

Программисты-проектировщики

Программисты стали первыми, кто «добровольно» вызвался сразиться с проблемами неискушенных в технологиях пользователей. Не имело значения, что у них не было для этого подходящих инструментов и наблюдались значительные различия в культуре, – другие кандидаты все равно отсутствовали. Программистам, будто случайным свидетелям катастрофы, было поручено оказать «первую помощь» интерфейсам только потому, что им «посчастливилось» работать в смежной области. Невозможно представить программиста, не обладающего энтузиазмом и гордостью в отношении своей деятельности и способности решать трудные задачи, потому их так привлекла сложность проектирования взаимодействия, где им пришлось приложить максимум усилий. Это породило весьма ядовитую шутку: «Проектирование у программистов – это те двадцать минут перед тем, как они начинают писать код».

На страницах этой книги я уже неоднократно давал понять, что все усилия программистов в области проектирования были изначально обречены. Как пишет По Бронсон, «отсутствие критики для них – лучшая похвала», так что свою производительность они оценивают невероятно высоко и многие из них уже довольно крепко сжились с ролью проектировщиков взаимодействия. Программисты, словно короли-безумцы, не желают сдавать захваченные владения, даже невзирая на то, что эта позиция невыгодна, неприятна, нежеланна и не пригодна для обороны.

В случае если вы занимаетесь программированием на профессиональном уровне, вы остаетесь программистом вне зависимости от того, каковы ваши способности в обучении, тестировании или проектировании. Как нельзя быть чуть-чуть беременной, так нельзя быть и чуть-чуть программистом.

Значительная часть разработчиков по-прежнему отказывается признавать наличие серьезной проблемы («пользователям просто нужно повышать свой уровень компьютерной грамотности»), однако существуют и те, кто ясно понимает, какое глубокое разочарование и большие финансовые потери влекут пляшущие медведи. Хорошей новостью является то, что вторая группа обретает все больше сторонников, а вместе с этим все больше компаний, занятых в сфере разработки ПО, изъявляют желание обратиться за помощью к сторонним специалистам.

На практике довольно многие программисты неплохо справляются с проектированием взаимодействия, а часть тех, кто не так в этом силен, признают свою слабость и стараются не браться за задачи проектирования. Есть лишь одно «но», и оно довольно значительно: в основе проектирования, выполняемого программистами, в большинстве случаев лежит их своеобразная индивидуальность, присущая виду *Хомо логикус*. В результате на свет являются неподобающие продукты, лишенные легкости в использовании, которые могут по достоинству оценить лишь такие же программисты.

Откуда вам это известно?

Многим юзабилити-специалистам свойственно полагать, что все удобство программы оценить невозможно, пока не будут проведены соответствующие тесты. Оттого они постоянно задают вопрос: «Откуда вам это известно?» Но тут я заметил одну очень любопытную вещь: задавая этот вопрос, они не стремятся изобразить адвоката дьявола, а спрашивают лишь потому, что не в состоянии отличить, является ли данное взаимодействие спроектированным качественно.

Мне доводилось работать как минимум с четырьмя крупными компаниями, каждая из которых плотно взаимодействовала с профессиональными специалистами по юзабилити, поскольку приняла решение вложить в юзабилити часть денежных средств. Они привлекли экспертов для создания лабораторий, организации исследований, поиска потенциально проблемных областей и выдвижения гипотез, каким образом можно улучшить ситуацию. Программисты добросовестно изменяли программы на основании полученных данных, *но результат практически не менялся*. Программистам лишь приходилось выполнять дополнительную работу и тратить на это больше времени. Спустя несколько таких итераций программисты просто опустили руки, аналогично поступили и многие руководители. К ним

наконец-то пришло осознание, что этот процесс требует значительных денежных и временных затрат, однако самую главную проблему он не решает.

На помощь проектировщикам взаимодействия приходит их опыт, специализированная подготовка и способность делать собственные выводы – все это значительно повышает точность их оценки ситуации. Они прибегают к помощи специальных методов, приемов, инструментов для определения проблемных зон и, кроме того, пользуются дополнительными источниками информации. Откуда врачу-рентгенологу известно после изучения рентгеновского снимка, что пациента нужно направить на операцию? Рентгеновские снимки так сложны для интерпретации, что врачам, не специализирующимся в данной области, невозможно и представить себе такое. В то время как врачи с особой подготовкой постоянно дают подобные оценки. Откуда известно судье, что подсудимый виновен? Как может инвестор на бирже знать, что пришла пора покупать? Возможно, эти профессионалы иногда принимают ошибочные решения, но в целом их деятельность основывается вовсе не на случайных предположениях.

Мне доводилось наблюдать, как солидные юзабилити-специалисты попадают в «молоко». Они тратили много времени на разработку самых изощренных тестов для отслеживания отдельных реакций пользователя на интерфейс, а после скрупулезно изучали таблицы результатов, пытаясь выявить недостатки программ. После того как с помощью их точных научных методов удавалось обнаружить проблемные зоны, они впадали в рассуждения, больше присущие любителям: «Так, думаю, эту кнопку можно перенести вот в то диалоговое окно» или «Полагаю, пользователю будет удобнее, если добавить кнопку вот здесь».

Если просто ответить «Не знаю», такой вариант могут принять, а вот попытки угадать будут обречены на провал. Хуже того, стоит вам начать гадать, глядя в сторону, программисты – те, кто кровно заинтересован в процессе, – спишут вас со счетов, как шарлатана, не удостоив и словом.

Руководства по стилю

В 1980-е годы дизайнер Шелли Ивенсон и ученый Джон Райнфран плотно сотрудничали на базе исследовательского центра компании *Xerox* в Пало-Альто, и это взаимодействие положило начало некоторым важным идеям из области визуальных коммуникаций. Результатом стало создание совместимого визуального словаря, который получил название «язык визуального дизайна» и применялся для всех фотокопировальных устройств *Xerox*: зеленым цветом на аппаратах обозначалась область для загрузки оригиналов документов, синим – область загрузки чистой бумаги, красным – область выдачи копий документов и обслуживания устройств. Аналогичные нетекстовые подсказки оказываются достаточно полезными в интерфейсах с высоким уровнем когнитивного сопротивления и описываются в документах, называемых «руководствами по стилю», которые представляют собой сборники с примерами и предложениями по использованию того или иного стилевого оформления.

Для многих разработчиков программного обеспечения, а также руководителей разработки, уставших от проблем с пользовательским взаимодействием, было бы огромным облегчением иметь под рукой подобное руководство, которое рассказало бы им, какой интерфейс нужен их продуктам. У многих компаний такие руководства по стилю разработаны для внутренних нужд по созданию программ, а некоторые поставщики программного обеспечения предлагают такие руководства независимым разработчикам, которые пишут совместимые программы.

Несмотря на то что руководства по стилю обычно оказываются полезны, они все же не решают проблем целеориентированного проектирования. Здесь для каждого случая должно быть свое решение. В зависимости от различных целей пользователи используют разные приложения, а каждое пользовательское взаимодействие продукта должно помогать выполнять свои цели. Общий визуальный язык и принципы последовательного управления могут помочь, но сами по себе они проблему не решают.

Конфликт интересов

Если бы Билл Гейтс публично потребовал от всех компаний, кроме *Microsoft*, прекратить внедрять проектирование взаимодействия, все они просто освистали бы его и выгнали со сцены. Тем не менее руководство по стилю интерфейсов *Microsoft* (Microsoft UI Style Guide) делает

именно это, и притом является одним из самых сильных конкурентных преимуществ *Microsoft* в индустрии.

И *Microsoft*, и *Apple* предлагают свои руководства по стилю к продаже и позиционируют их как невероятно содержательные и полезные. На первый взгляд кажется, что эти компании представляют собой весьма авторитетные источники. Однако интересы поставщиков платформенных решений вступают в коварный конфликт с интересами разработчиков, так что их посылам не стоит всецело доверять.

В намерениях обеих этих компаний присутствует некая форма скрытого принуждения к соблюдению установленных норм. Если независимый разработчик программ не будет соблюдать предписания руководства по стилю, его продукт не получит отметку о «совместимости с платформой», а значит, лишится важного маркетингового преимущества. Потому большая часть из тех, кто занимается разработкой ПО, готова соблюдать предписания, установленные поставщиками платформ.

Таким образом, устанавливая требования к продуктам независимых создателей программ, эти компании незримо подавляют внедрение инноваций со стороны сообщества разработчиков.

Между тем самим разработчикам платформенных решений позволено проводить эксперименты, развиваться и внедрять инновации, сколько им будет угодно. Собственными руководствами по стилям они вольны пренебрегать. Конечно же, никто столь часто и таким возмутительным образом не нарушает предписания в этих руководствах, как те же *Microsoft* и *Apple*.

Я отнюдь не призываю напрочь игнорировать рекомендации по стилю и оставлять интерфейсы на волю случая. Я только хочу сказать, что к подобным руководствам нужно относиться так же, как сенатор относится к лоббисту, а не как автомобилист к сотруднику дорожной инспекции. Представитель власти знает, что лоббист преследует собственные интересы, а вовсе не является беспристрастной третьей стороной.

Фокус-группы

Многие отрасли осознали ценность фокус-групп как способа изучить, что пользователи приветствуют, а что нет в различных продуктах. Однако при всей пользе фокус-групп для получения информации о том, что думают потребители о тех или иных продуктах, в индустрии разработки программного обеспечения такие фокус-группы приносят больше хлопот, чем пользы. Самая большая проблема элементарно заключается в том, что большинство пользователей, даже тех, кто использует программы профессионально, не осведомлены, что представляет собой программное обеспечение в целом, каким оно может быть, а каким нет. Потому каждая опция, которую запрашивает участник фокус-группы, является следствием не слишком дальновидного мышления. То есть пользователь просит то, что по его личному мнению может быть подходящим, возможным и разумным. Попросить о чем-то заведомо малопригодном, невозможном или неразумном – значит добровольно выставить себя на посмешище, а люди этого не любят.

Ученые Насс и Ривз из Университета Стэнфорда занимались изучением того, как люди реагируют при взаимодействии с компьютером, и получили неопровержимые доказательства, что полагаться на оценку пользователями собственных реакций нельзя – она недостоверна. Вот что пишут ученые: «В основе использования многих широко распространенных методов, в частности фокус-групп, лежит предположение, что людям свойственна интроспекция в отношении [интерактивных] взаимодействий. Мы полагаем, что это предположение чаще всего оказывается неверным».

По словам Ларри Кили, «пользователи откажутся от новых идей, если им их предложить». А потому фокус-группы – это довольно сомнительный инструмент при внедрении значимых инновационных решений. Сегодня большинство программных продуктов содержат существенную долю инноваций, а потому использовать фокус-группы бессмысленно.

Для некоторых групп продуктов фокус-группы могут оказаться эффективными, но применять их для достоверной оценки продуктов с высоким уровнем когнитивного сопротивления было бы ошибкой.

Визуальное проектирование

В книге «Алан Купер об интерфейсе. Основы проектирования взаимодействия» я продемонстрировал, что графический пользовательский интерфейс (Graphical User Interface, GUI) стал преобладающим способом взаимодействия с компьютером далеко не из-за своей графической природы. Новые графические интерфейсы возобладали над предшествовавшими им зелеными экранами скорее благодаря жестко ограниченному набору возможных взаимодействий, на основе которого они и были созданы. Хорошее визуальное проектирование, безусловно, может значительно повлиять на качество любого интерфейса, но многие специалисты индустрии по-прежнему наделяют его той ценностью, которой оно на самом деле не обладает.

Как-то мне довелось быть в составе жюри в конкурсе по проектированию и конструированию прикладных программ для внутреннего пользования в компаниях[37]. Одним из победителей стала программа для управления продажей билетов, созданная для ежегодного съезда авиалюбителей в Висконсине. Сердцем этой системы был кассовый терминал – устройство, намеренно лишенное графического интерфейса. В нем был лишь текстовый дисплей, обладавший особой прочностью, простотой и легкостью в восприятии информации для пользователя. Тем не менее программа совершенно справедливо оказалась в числе лидеров, потому что в процессе проектирования разработчики тщательно исследовали специфические потребности группы добровольцев, задействованных в продаже билетов на съезде. Работа этих добровольцев была критически важной, но вместе с тем простой, ее требовалось выполнять очень быстро и с минимумом подготовки. Графические интерфейсы невероятно хороши в тех случаях, когда руководители хотят видеть полную картину положения дел в компании, но операторы кассовых терминалов не нуждаются в этом, потому что каждый следующий покупатель в очереди не имеет ничего общего с другими покупателями. Так что отображение всей картины в целом не входило в задачи программы. Установки простейшего текстового дисплея было вполне достаточно, чтобы продукт мог стать обладателем награды. Однако многие практикующие специалисты упускают этот урок из виду.

Одной из характеристик GUI является способность отображать полноцветную растровую графику. Так становится возможным создавать визуально насыщенные интерфейсы программ, такие же яркие, как графика в игре Myst. Поэтому к вашим услугам множество графических дизайнеров и художников, которые с готовностью покроют лик вашей программы привлекательной маской растровой графики. Тем не менее графические дизайнеры едва ли задумаются о пользовательском взаимодействии, которое должно лежать в основе этих действий.

Программа на изображении ниже по сути своей представляет собой бесполезную мишуру с приятным глазу интерфейсом, выдаваемую вместе с новыми компьютерами совершенно бесплатно, что, в общем-то, и определяет ее ценность. Кажется, она нужна для телефонной связи, а может, для запуска CD-ROM – я точно не знаю. Ее интерфейс, бесспорно, может показаться вам красивым, в особенности если вы технофил с любовью ко всяческим гаджетам, однако совершенно невозможно постичь, как им пользоваться. Такие примеры мы называем «приукрасить мертвеца». Разработчики взяли интерфейс, которым невозможно пользоваться из-за глубинных изъянов поведенческого проектирования, и облачили его в обольстительные одежды.

Складывается впечатление, что поставщикам технических устройств такой подход кажется особенно привлекательным – напомню, что программа бесплатно предоставлялась вместе с моим новым компьютером. По моим подозрениям, все дело в том, что интерфейс в представлении разработчиков является самым простым способом выразить все великолепие аппаратных возможностей.

Нам часто встречаются продукты, которые *кажутся* невероятно привлекательными – их внешний вид весьма хорош, – однако нельзя сказать того же об их функциональности и пользовательском взаимодействии, – они оказываются неадекватны. И причина кроется вовсе не в отсутствии проектирования, а в том, что оно выполнялось графическим дизайнером, уделяющим внимание эстетике, а не проектировщиком взаимодействия, который обладает рядом инструментов для предупреждения появления высокого уровня когнитивного сопротивления.

Методы промышленного дизайна

Еще одна профессиональная отрасль, к которой нередко обращаются за получением экспертного мнения, – это промышленный дизайн. Эта отрасль представляет собой давнюю, устоявшуюся практику профессионального создания трехмерных объектов, которые хорошо адаптированы под ваши глаза, тело и в особенности под ваши руки. Если говорить в общем, то промышленные дизайнеры делают отличную работу, но в применении к программам их преимущества оказываются скорее недостатками. Они обучены создавать элементы управления, которых можно легко коснуться физически, рассмотреть их и производить с ними различные манипуляции, – кнопки, ручки, регуляторы. Но они не имеют специальной подготовки для того, чтобы должным образом справляться с когнитивным сопротивлением или взаимодействовать с разработчиками программ. Как в случае с системой бесключевого автомобильного доступа, описанной в главе 2 «Когнитивное сопротивление», пользователь легко сможет опознать кнопку даже на ощупь. Что с ними делать физически – нам совершенно ясно, а вот их логическое назначение (метафункция) по-прежнему вызывает массу вопросов.

Если посмотреть на пять пультов дистанционного управления, лежащих на моем кофейном столике, можно заметить, что каждый из них в отдельности сделан весьма неплохо, но все они в целом делают мой домашний центр развлечений абсолютно непригодным к использованию. Несмотря на удобные изгибы корпуса и приятный внешний вид этих пультов, вы окажетесь в безнадежном положении, если попытаетесь переключить канал или отключить звук в неосвещенной комнате. Те дизайнеры, которые проектировали эти пульты, выполнили все требования поставщиков оборудования, но упустили из виду требования пользователей к качественному взаимодействию.

Несложно догадаться, почему менеджеры по продукту ошибочно принимают промышленный дизайн за проектирование взаимодействия. Промышленные дизайнеры тоже работают с «интерфейсом», посредством которого пользователи взаимодействуют с высокотехнологичными артефактами, облегчая использование последних. Однако тот факт, что кнопки можно быстро найти и нажать, вовсе не означает, что пользователь так же быстро догадается, какую именно из этих кнопок нажимать. И эта проблема лежит в области когнитивного сопротивления, а не промышленного дизайна.

Новая первоклассная технология

Остался еще один претендент на престол проектирования взаимодействия – и это технология собственной персоной. Такую мнимую панацею, в частности, усиленно пропагандирует *Microsoft*, утверждая, что интерфейсы станут невероятно удобными, как только будут успешно реализованы технологии распознавания речи и рукописного ввода. Мне такое видение кажется весьма нелепым. С появлением каждой новой технологии у пользователей лишь прибавляется шансов испортить себе настроение, только пользуясь уже более быстрыми и мощными системами.

Ключом к созданию более качественного пользовательского взаимодействия является снижение уровня неопределенности между компьютером и человеком. Технологиям обработки естественного языка никогда не достичь этого, поскольку в человеческой речи кроется слишком много разных смыслов. Значительная доля наших разговоров построена на использовании

нюансов слов, жестов и интонаций, так что пройдет еще год-другой, прежде чем компьютеры научатся распознавать слова, и десятилетия – если это вообще возможно, – прежде чем они смогут достоверно извлекать из них смысл.

Несомненно, технологии распознавания речи могут оказаться полезными в некоторых продуктах. Однако, на мой взгляд, было бы слишком опрометчиво оптимистично полагать, что с приходом новой технологии мы вдруг окажемся спасены, хотя этого до сих пор не случилось. Любые технологии в целом требуют законченных решений по проектированию взаимодействия для конечных пользователей, вне зависимости от того, какой набор этих технологий применяется.

Итерации

В сфере разработки программного обеспечения существует широко распространенная истина, что получить качественное пользовательское взаимодействие можно только методом итераций. Широкому хождению этой идеи способствовала приверженность юзабилити-тестированию в большинстве университетов, а также во многих крупных компаниях, к которым относится и *Microsoft*. Безусловно, итерации можно считать важной частью хорошего процесса проектирования, – повторяйте до тех пор, пока не получится. Тем не менее некоторые разработчики превратно истолковали суть этой идеи, приняв ее как возможность пренебречь проектированием и в ходе тех же итераций просто-напросто сделать несколько случайных выстрелов наугад.

В 1986 году компания *Microsoft* наскоро выпустила на рынок самую первую версию Windows, которая оказалась настолько жалкой, что справедливо стала посмешищем всей индустрии. Полгода спустя компания представила версию 1.03, в которой были исправлены некоторые мелкие погрешности. Еще через год *Microsoft* выпустила версию 1.1, а чуть позже – версию 2.0[38]. В ходе каждой следующей итерации разработчики продукта пытались справиться с проблемами, возникшими в предыдущей версии. Наконец, спустя четыре года от начала выпуска первой версии компанией *Microsoft* была представлена Windows 3.0, и только тогда смех в индустрии утих. Не так много компаний в этой индустрии обладают такими финансовыми возможностями или, может быть, стойкостью, которые позволили бы им выносить публичное унижение в течение четырех лет и в итоге все же создать оптимальное решение. Побочным эффектом этой ситуации является то, что вся индустрия следит, как признанный лидер делает неуверенные шаги, словно незрячий, до тех пор, пока все наконец-то не встанет на свои места. Как следствие, индустрия принимает очевидно абсурдное решение считать такой подход единственно верным.

Тем не менее для *Microsoft* создание такого множества промежуточных версий стало весьма затратным предприятием. Если бы компания пошла несколько иным путем и довела продукт до качества Windows 3.0, не выводя на рынок пару промежуточных версий, ей удалось бы сэкономить порядка нескольких миллионов долларов на разработке и поддержке продукта и заработать дополнительные миллионы за счет продаж на более раннем этапе жизненного цикла продукта (к слову сказать, пользователи бы тоже сэкономили миллиарды долларов и избежали многих головных болей). Принимая как данность неизбежный выпуск множественных версий продукта, разработчики и компании просто-напросто вынуждают здравый смысл капитулировать и при этом несут огромные затраты.

В основе стратегии *Microsoft* лежит примитивная тактика «войны на истощение». Согласно военной терминологии, эта тактика означает, что, если силы врага равны или уступают вашим, количество ваших солдат и вооружения так велико, что вы будете «играть на понижение» за счет обоюдных жертв до тех пор, пока противник больше не сможет защищаться. Если говорить языком программного обеспечения, это значит, что нужно выводить на рынок плохой продукт – настоящего пляшущего медведя, а затем выслушивать стоны и жалобы пользователей. Затем дорабатывать то, что им не подошло, и выпускать обновления. Спустя три или четыре таких версии открытые раны пользователей затянутся, а качество продукта достигнет некоторого допустимого минимума благодаря широте функциональных возможностей, после чего показатели качества расти перестанут. Метод итераций никогда не приводит к созданию великих продуктов.

Подобная стратегия, основанная на истощении противника, является не только затратной, но также требует огромного количества времени. Более того, она невероятно отвратительна, поскольку унижает человеческое достоинство пользователей, которые применяют компьютерные технологии. К сожалению, для *Microsoft* эта стратегия оказалась неплохой находкой. Раз за разом компания выводит на рынок «полуфабрикаты» – непродуманные, некачественные в отношении проектирования и конструирования продукты, вызывая насмешки обозревателей индустрии – как приверженных компании, так и беспристрастных. Но несмотря на издевки экспертов, *Microsoft*продолжает поддержку своих изначальных версий продуктов выпусками вторых, третьих, четвертых, пятых и одиннадцатых версий. В результате продукты этой компании, среди которых Windows, ActiveX, Word, Access, Windows NT и множество других, превратились в гигантов соответствующих рынков.

Стратегия на основе истощения работает только в тех случаях, когда компания обладает нерушимым брендом, уймой времени, самообладанием игрока в покер и обширными финансовыми возможностями. До настоящего времени ни одна другая фигура в индустрии компьютеров не продемонстрировала наличие подобных качеств на уровне, достойном *Microsoft*.

Настоящая проблема, которую повлек столь грандиозный успех *Microsoft*, – это попытки многих компаний меньшего размера пройти тем же путем, повторяя те же стратегические и тактические приемы «войны на истощение». Как показала практика компании *Netscape* с ее веб-браузером, такой подход в долгосрочной перспективе нередко оказывается провальным, тем не менее все это лишь развивает традицию унижать конечных пользователей.

Победить игрока, который использует стратегию на основе истощения, вполне возможно, но для этого нужно применить иной подход. Кем бы вы ни были, денег у *Microsoft* в конечном счете больше. Вместо этого нужно целенаправленно наносить удары по самому слабому месту *Microsoft*– по процессу разработки этой компании, в котором этап программирования идет перед этапом проектирования взаимодействия. *Microsoft* становится вдвойне уязвимой из-за того, что задачами, связанными с дизайном и проектированием, в компании занимается слишком большое количество так называемых проектировщиков. Как было показано в выдержках из книги Фреда Муди в главе 8 «Вымирающая культура», *Microsoft* уже зарекомендовала себя как компанию с неэффективным подходом к проектированию постфактум. Каждый, кто готов создавать настоящие решения по проектированию взаимодействия, сможет победить *Microsoft* в этом отношении.

13. Управляемый процесс

На мой взгляд, большая часть из тех руководителей, что заняты в сфере разработки программных продуктов и устройств на их основе, на самом деле не имеют ясного представления о том, как выявлять лучшие, наиболее успешные продукты и даже как их создавать. В результате такой неопределенности они идут на поводу у своих страхов, а это, в свою очередь, все равно что играть с огнем. Их действия быстры и точны, но они не контролируют ситуацию, а потому рискуют обжечься, стоит им отвлечься лишь на мгновение. В этой главе я на практике объясню, с какой дилеммой сталкивается каждый технический руководитель, и покажу, как проектирование поможет совладать с огнем.

Кто на самом деле обладает всей властью?

Как определить, чей совет принять во внимание, а чьим пренебречь? Мне доводилось видеть руководителей, чьи действия напоминали поведение собаки на оживленном перекрестке. Очутившись в центре потока автомобилей, такие собаки поднимают яростный лай и пытаются бежать за каждой проезжающей мимо машиной, во все стороны сразу. Так и в компаниях – высшее руководство требует: «Сделайте программу похожей на Outlook 98». Маркетологи твердят: «Нам нужно не хуже, чем у конкурентов». Отдел продаж добавляет: «Тому покупателю нужна вот такая функция». Программисты заявляют: «Программа должна быть совместима с предыдущей версией». Кому же верить?

Руководители, ответственные за разработку продукта, изо всех сил пытаются угодить всем сторонам этого процесса. В таких случаях наибольшая власть принадлежит программистам – в их руках программный код, а потому именно их требованиям отдается наивысший приоритет. Тем не менее остается еще одна группа, чьи потребности также кажутся наиболее важными, – потребители. Ведь какие бы предложения ни вносили прочие стороны процесса, в конечном

счете именно *потребитель – тот, у кого в руках деньги*. Ни один человек из бизнеса не будет пренебрегать этим фактом.

Смертельный номер: ведомые потребителем

Соглашаясь получать плату за свой продукт, вы встаете на путь превращения в компанию, «ведомую потребителем». Идея кажется заманчивой и широко применяется, тем не менее следовать ей – ошибочное решение. Так вы оказываетесь сразу втянуты в опасную игру с огнем. В 1980-е годы компания *IBM* была невероятно горда тем, что клиенты указывают ей путь, пока они не привели ее к пропасти. В то время *IBM* фактически полностью владела всем рынком компьютеров, ее охват был куда более широк, чем сейчас у *Microsoft*, однако сегодня *IBM* – всего лишь один из многих игроков этого рынка, ее масштабы все еще велики, но она утратила свои лидерские позиции.

Новые компании, как правило, создают свой первый продукт, закладывая в его основу какое-либо технологическое новшество. Они проектируют этот продукт, руководствуясь своими внутренними представлениями, как следует делать подобные вещи. На этой стадии клиенты еще не слишком лояльны, а потому их пожелания относительно возможностей продукта выглядят крайне бессвязно. Однако стоит финальной версии продукта выйти на рынок, как интерес потребителей возрастает в разы, поскольку они вкладывают в этот продукт свое время и силы. Вполне ожидаемо, что они начинают направлять компании свои предложения по изменениям и дополнениям.

Однако между тем, чтобы просто *прислушиваться* к мнению клиентов, и тем, чтобы ему слепо *следовать,* существует огромная разница. Прислушиваться – это разумное решение. Это означает, что вы пропускаете всю полученную информацию через себя. Слепо следовать указаниям клиентов – плохое решение. Это значит, что вы просто выполняете все, чего от вас захочет потребитель. В таком случае уже не вы ведете игру с огнем, а он начинает играть с вами.

Стоит производителю продукта позволить потребителям диктовать свои требования относительно возможностей продукта, как случается весьма серьезная, но поначалу совершенно незаметная перемена. Из *производителя* продуктов, изобретающего собственные решения для продажи потребителям, компания превращается в *обслуживающую* организацию, которая работает по заказам клиентов. Каждый сотрудник компании улавливает этот мельчайший сдвиг и реагирует на это соответствующим образом – начиная превозносить интересы потребителей выше, чем чьи-либо другие.

Множество производителей корпоративного ПО, таких как Oracle и SAP, кому довелось пройти через взрывной рост в начале 1990-х, когда устаревшие программы для мейнфреймов были заменены современными клиент-серверными архитектурами, сегодня столкнулись с тем же «ночным кошмаром» *IBM*, идя на поводу у клиентов. Представив широкой публике свою новую технологию, эти компании, производящие продукты в так называемой категории ERP (Enterprise Resource Planning, планирование ресурсов предприятия), начали слушать, что скажут потребители. Затем они стали дополнять продукты теми опциями, что запрашивали клиенты, совершенно не согласовывая их с собственными глобальными целями и долгосрочной стратегией.

От некоторых руководителей мне приходилось слышать, что в их продукты не вносятся никакие изменения и дополнения, пока не поступит такой запрос от потребителей. При этом у каждого из их клиентов организация процессов построена несколько иначе, а потому от ERP-продуктов они требуют таких возможностей и опций, которые соответствуют именно их системе действий. Компаниям, слепо идущим на поводу у потребителей, приходится соглашаться, однако они ошибочно принимают попытку угодить за стремление быть полезными.

У одной компании могут быть десятки и сотни подобных потребителей. Если она будет пытаться угодить каждому из них (или даже только самым крупным клиентам), кто должен урегулировать их противоречивые запросы?

У многих знакомых мне руководителей из сферы высоких технологий имеется собственный опыт разработки продуктов, а часть из них – бывшие программисты. Они занимают такие должности как минимум по той причине, что весьма хорошо понимают программистов и симпатизируют им. Как я уже говорил в главах 7 «Homo logicus» и 8 «Вымирающая культура»,

программисты считают опции и возможности выходом во всех ситуациях. Стоит клиенту появиться со списком требований в одной руке и денежными средствами в другой, – у технического руководителя не остается сил сопротивляться. Еще одной причиной, почему так много компаний-разработчиков идут на поводу у списка опций, является тот факт, что они играют с огнем, а дедлайны являются гарантией того, что темпы этой игры будут стремительно нарастать.

Концептуальная целостность – ключевая компетенция

Как только вы принимаете денежные средства от клиента, вы вручаете ему бразды правления, позволяя направлять процесс разработки в вашей компании. Деньги клиента важны, однако, помимо этого, у него нет как минимум двух главнейших качеств, которые есть у вас: 1) понимание долгосрочных целей и 2) знание того, как нужно проектировать продукт.

Те продукты, которые создаются под запросы клиентов, получаются весьма невразумительными. В них недостает того, что гуру программного обеспечения Фредерик Брукс описывает как «концептуальная целостность», – некое видение цели продукта, которое, как продолжает далее Брукс, и является наиболее важным ингредиентом успеха. Если такой концептуальной целостности нет, могут произойти две вещи: процесс проектирования вашего продукта оказывается под контролем клиентов, а вы складываете с себя полномочия за это. Клиенты, вне зависимости от того, насколько благородны их помыслы, не могут взглянуть на ваш продукт как на единое, концептуальное целое. Способность к четкому видению ситуации – это ключевая компетенция, а многие компании из категории ваших клиентов с трудом могут сосредоточиться на *собственном* бизнесе, не говоря уж о вашем. Поэтому несмотря на то, что клиенты выдают вам противоречивые указания, они рассчитывают, что вы *сами* сумеете правильно определить, чему из этого следовать.

Если вы идете на поводу у клиентов при разработке продукта, то от версии к версии он лишь бесконтрольно мутирует, в то время как должен развиваться последовательно. В итоге продукт выглядит как набор бессвязных элементов и опций, добавленных случайным образом, отчего становится похож, как выразился Джон Зикер, на «собачий завтрак». Каждому пользователю приходится пробираться сквозь дебри возможностей продукта, выискивая нужные и избегая неподходящие, при этом абсолютно все клиенты замечают, что с каждой новой версией использовать продукт становится все труднее. Программные продукты некоторых достаточно известных компаний настолько сложны, что пользователю требуется многомесячное обучение, чтобы справиться даже с самой простой задачей. На этой волне возникают целые отдельные компании, которые занимаются обучением, установкой, настройкой и сопровождением этих монструозных программных решений. И хотя клиенты будут покупать подобные «собачьи завтраки», влюбиться в такой продукт невозможно. Он не обладает желанностью, а это, как я упоминал в главе 5 «Нелояльность клиентов», делает продукт – и вас вместе с ним – уязвимым для конкурентов.

Фаустов договор

Такую трансформацию компании из организации с четким видением продукта в организацию, ведомую клиентами, можно трактовать как переход из категории производителей в категорию обслуживающих компаний. В великолепной книге Дэвида Майстера *Managing the Professional Service Firm*[39] («Управление фирмой, оказывающей профессиональные услуги») рассказывается о проблеме следования запросам клиентов в ином контексте: с точки зрения поставщика услуг, оказывающего консультации. Конечно, бизнес по предоставлению услуг имеет существенные отличия, потому некоторые понятия в книге описываются другими словами. Так, продажа *экспертности* в умении решать проблемы в ней противопоставляется продаже прошлого *опыта*. Дэвид Майстер использует для обозначения этих понятий слова «ум» (для экспертности) и «мудрость» (для опыта).

Ум продавать непросто. Тот, кто обращается к вам за консультацией, надеясь на ваш ум, вынужден оказать вам большой кредит доверия, потому что ожидает от вас выполнения таких задач, в которых вы еще не доказали свою компетентность. Куда как проще продавать мудрость.

Ваш потенциальный клиент может убедиться, что раз вы прежде уже решали такие задачи, значит, сумеете сделать это снова.

Большинство консультантов начинают с того, что продают свой ум коллегам, то есть тем людям, с которыми у них уже выстроены доверительные отношения. Как только консультанту удается решить проблему клиента, начинает формироваться репутация и клиентов становится больше. Чем больше клиентов, тем менее они знакомы с консультантом и тем меньшего доверия можно от них ожидать. Потому неизбежно появятся задачи, которые потребуют действительного опыта. Ведь именно он привлек вам нового клиента, а подобные клиенты обычно дают такие проверочные задания непроверенным консультантам.

Когда нарабатывается репутация, увеличивается и число клиентов, которых привлекла мудрость консультанта, и в итоге к нему приходит понимание, что за опыт дают больше денег и достаются они легче. Ведь он фактически решает все те же проблемы, которые до этого решал множество раз.

По мере того как бизнес консультанта превращается из продажи ума в продажу опыта, начинают исчезать те самые качества, благодаря которым консультанта так ценили. Он утрачивает сноровку. Теперь его услуги – это уже не виртуозное решение интересных проблем, а монотонное выполнение привычных задач. Его популярность падает, а существующие клиенты дают ему задачи все более и более низкого уровня, что только уничижает его. Более того, его клиенты начинают обращаться за помощью к другим консультантам, которые ушли далеко вперед и лучше используют свой ум.

Это тот же самый смертельный номер для тех, кто ведом клиентами, но теперь с точки зрения предоставления услуг.

Мораль этих историй такова, что компании, ведомые клиентами, получают легкие деньги в краткосрочной перспективе, но лишают себя возможностей для роста и перспективного будущего. Они отрекаются от своего пути лидера.

В этой тайной игре участвуют все стороны. Потребителям такое положение вещей вполне подходит. Новый клиент говорит вам: «Не могли бы вы добавить вот эту опцию в ваш продукт? Тогда бы я его купил». Это небольшая проверка, посредством которой можно узнать, легко ли с вами договориться. Подразделения, ответственные за продажи, прилагают немало усилий для совершения таких крупных сделок, а потому создается впечатление, что добавить одну маленькую опцию – крайне незначительная затрата в обмен на установление отношений с новым клиентом. Потенциальная прибыль обладает невероятно притягательной силой.

Решение, которое предлагает Дэвид Майстер в своей книге, до простого очевидное: чаще брать проекты, где требуется задействовать ум. Если говорить об оказании услуг, то здесь потребуется убедить клиентов, которых привлек ваш опыт, давать вам больше задач, требующих использования ума. Далее в книге Дэвид в деталях описывает, как это сделать. Он говорит, что для этого потребуется отказаться от легких денег за проекты, где нужен только опыт, и отдавать приоритет более сложным и менее доходным проектам, где понадобится применить ум. Если перевести этот способ решения проблемы в плоскость разработки программного обеспечения, можно понять, что запросы потребителей – это проекты, требующие только опыта, в то время как задачи для ума – это внутренние инициативы компании. Иначе говоря, именно на вас как руководителе, ответственном за разработку продукта, лежит задача оставаться на передовой и избегать смертельных трюков, в которые вас втягивают потребители. Нужно делать то, что вы делали, когда только стартовали, – искать все ответы внутри себя.

Это значит, что вам нужно составить прогнозы на более длительный период, принять на себя ответственность, управлять временем и все контролировать.

Прогнозирование

Чтобы выдержать натиск конкурентов, нужно подвергать краткосрочную прибыль оценке в долгосрочной перспективе. Вам необходимо быть абсолютно уверенным, что ваши сотрудники понимают: фокусироваться исключительно на краткосрочной прибыли – это все равно что запустить бомбу с часовым механизмом. По возможности избегайте такого подхода, даже несмотря на затраты в краткосрочном периоде.

Следование долгосрочным прогнозам означает, что вам придется отказаться от некоторых весьма привлекательных сделок. Сделать это бывает непросто, однако это совершенно необходимо для того, чтобы удержаться на плаву в будущем. Как говорит мой опыт, на самом деле такие сделки в действительности редко теряются. Если у вас хватает уверенности отказать клиенту, протягивающему деньги, это с большой долей вероятности лишь повысит уровень его доверия к вам, а кроме того, он произведет переоценку своих требований. Тем не менее для таких действий нужна сильная воля.

Принятие ответственности

Чем раньше вы найдете баланс, тем лучше. Вы не можете думать: «Я воспользуюсь краткосрочным планированием только первые два-три года, а потом составлю долгосрочный прогноз». Вам нужно найти между ними баланс в самый первый день. Всегда можно отложить краткосрочные прогнозы, а вот с долгосрочными так поступать нельзя.

Все дело в принятой корпоративной культуре. Конечно, в сложившиеся подходы краткосрочного планирования довольно трудно сразу встроить долгосрочные прогнозы. Кажется довольно рискованным отступиться от прибыльного угождения запросам клиентов, но это приведет вас в пропасть. Принимая огонь на себя, помните, что поступаете правильно.

Управление временем

Политика в отношении программных продуктов у многих высокотехнологичных компаний такова, что они выпускают новую версию продукта каждый год. Часть из этих компаний выпускает даже по несколько версий в год. Это значит, что жизненный цикл продукта, в разработке которого занята основная масса программистов, должен пройти стадии идеи, проектирования, программирования, тестирования и вывода на рынок в рамках одного года. Этого времени недостаточно для внесения сколько-нибудь значимых инноваций во внешний вид продукта, потому для большинства компаний характерно проводить стадии проектирования и программирования параллельно. Как я уже в деталях пояснял, при наложении программирования на проектирование в итоге получается только лишь программирование.

Контроль над процессом

Пожалуй, самое важное для руководителя высокотехнологичными разработками – это не позволить огню завладеть всем. Нужно смело вступить в схватку с огнем и взять ситуацию в свои руки, несмотря на возможный ущерб. Если вам удастся его одолеть, вы сможете приступить к перестраиванию процесса, с тем чтобы оптимально сбалансировать задачи, требующие ума и опыта, и чтобы в будущем эти преимущества остались за вами.

В поисках краеугольного камня

Многим компаниям свойственно тщательное планирование и анализ финансовых и операционных аспектов бизнеса. Когда же речь заходит о продукте, им кажется, что составление списка опций и есть его скрупулезное планирование, однако это далеко не так. Процесс разработки программных продуктов является слишком тяжелым, затратным и сложным, чтобы можно было обойтись без такого планирования и анализа перспектив. Применительно к программному обеспечению «скрупулезное планирование» означает только лишь проектирование взаимодействия, которым, как мы уже выяснили, нередко просто пренебрегают.

От целеориентированного проектирования есть и дополнительные выгоды, одна из которых – набор персон, то есть список определенных типов пользователей. Документ со списком персон оказывается невероятно полезным, когда нужно понимать, как реагировать на тот или иной запрос пользователя в отношении набора опций. Первым делом нужно определить, какой из персон эта опция будет нужна, а затем понять, входит ли данная персона в список ключевых образов. Если ответ положительный, опцию можно принять во внимание. Если же нет, то добавление этой опции оставит вас далеко позади ваших планов и лидерских позиций, вне зависимости от количества денег, которые вы за нее выручите. Представьте, что к вам в офис

заявится клиент и пообещает 100 тысяч долларов, если вы избавитесь от своей бухгалтерской системы или сожжете шкафы с документами. Согласитесь ли вы на такую сделку?

Знать, где резать

Когда компания становится ведомой клиентами, это четкий признак того, что ее руководители, ответственные за разработку продукта, верят в миф о непредсказуемом рынке. На самом деле им неизвестно, полезна ли данная опция или нет, можно ли обойтись без нее или нет. Они просто-напросто передают контроль над ситуацией потребителям – ну а почему бы и нет? Ведь у них *самих* нет четкого мнения на этот счет. Когда клиент говорит что-то вроде: «А добавьте-ка к функционалу разводной гаечный ключ для левшей», руководитель разработки думает, что клиенту, должно быть, известно нечто неведомое другим. Руководителю кажется, что эта опция – именно то, что каким-то магическим образом способно обеспечить продукту головокружительный успех.

У этой медали есть и обратная сторона – руководителю разработки точно так же неизвестно, от каких опций отказаться. Когда сроки сжимаются по воле внешних обстоятельств, руководителю приходится жертвовать какими-либо опциями, только все дело в том, что он не может определить, какие из них жизненно необходимы, а какие служат лишь ни на что не влияющим внешним лоском.

Позволить людям, не имеющим отношения к проектированию, выбирать, какие опции урезать, – это все равно что позволить кому бы то ни было обрезать провода в самолете. Какой из проводов резать, выбирается наугад либо производится исходя из не относящихся к делу признаков – таких как, например, цвет изоляции провода или расстояние до пассажирского кресла. Но так можно задеть важные провода. Одно дело – случайно лишить освещения кресло под номером 22A и совсем другое – обрезать провода, обеспечивающие питание двигателя. Если же урезáть опции будут проектировщики, то они подойдут к этой задаче так же, как будет обрезáть провода создатель самолета: он не тронет те из них, что необходимы для нормального полета, а в первую очередь обесточит оборудование, жизненно важной необходимости в котором нет.

Производство фильмов

Процесс производства фильма обходится столь же непомерно дорого, как и разработка программного обеспечения. Киноиндустрия в Голливуде существует намного дольше, чем индустрия разработки ПО в Кремниевой долине, так что нам есть чему поучиться у создателей кинолент. По-настоящему затратная часть – это непосредственно съемка фильма. Камеры, сцены, технические специалисты, актеры – все это требует многих тысяч долларов за каждый съемочный день. Опытные создатели фильмов держат эту фазу производства под строгим контролем, заранее составляя четкие подробные планы. Закладывая денежные средства и время на подготовку детальных раскадровок и графиков съемок, они экономят гораздо больше времени и денег во время самого съемочного процесса.

Создание фильма обычно включает в себя три основные фазы: предпроизводство (препродакшн), производство и постпроизводство (постпродакшн). В фазе предпроизводства продюсеры фильма отвечают за подготовку и доработку сценария, разработку сцен и костюмов, кастинг актеров и съемочного персонала, привлечение денежных средств. Фаза производства – это блеск софитов, гул кинокамер, крики режиссеров, раздающих указания, и накал страстей, которые разыгрывают актеры. В фазе постпроизводства происходит монтаж киноленты, запись саундтрека и подготовка маркетинговой кампании. Эти три фазы достаточно близко соотносятся со стадиями разработки программного обеспечения.

Руководители, отвечающие за разработку программ, на стадии предпроизводства осуществляют проектирование продукта, нанимают на проект программистов и привлекают инвестиции. Стадия производства здесь – это «блеск» мониторов, «гул» компиляторов, крики руководителей, раздающих указания, и накал страстей программистов, генерирующих код. На стадии постпроизводства выполняется отладка кода, подготовка документации и разработка маркетинговой кампании.

На экономию времени требуется время

Примечательной особенностью такого тройственного процесса является тот факт, что задача фазы предпроизводства – *свести продолжительность фазы производства к минимуму*. Непосредственная съемка фильма обходится чрезвычайно дорого, а потому урезание съемочного процесса даже на два дня с лихвой компенсирует лишние недели пред- или постпроизводственного процесса. На сегодняшний момент предварительная подготовка к съемкам фильма на стадии предпроизводства может длиться год и более, на процесс производства отводится два-три месяца интенсивных съемок, а фаза постпроизводства занимает еще много долгих месяцев.

С усложнением фильмов в техническом плане (а что будет, если скрестить компьютер и фильм?) все больше и больше операций из фазы производства не могут осуществляться без полного детального планирования. Если по ходу съемок актер в главной роли должен сразиться на лазерных мечах с пришельцем, который будет создан с помощью компьютерной программы, такая драка должна быть воображаемой и сниматься на фоне синего экрана, а потому каждое действие актера – от мельчайших движений до направления взгляда – нужно тщательно продумать.

Создателям фильмов прекрасно известно, что другого шанса сделать все как нужно у них не будет, а потому они относятся к фазе предпроизводства со всей серьезностью. Если же говорить об индустрии разработки программного обеспечения, то здесь дело обстоит иначе: многие руководители уверены, что они смогут запросто все исправить при выпуске следующей версии программы, вследствие чего важность тщательного планирования несколько принижается. В результате расплачиваться за подобное допущение приходится слишком дорого.

Современное программное обеспечение не менее сложно в разработке, чем фильмы, однако в большинстве процессов проектирования этот факт попросту игнорируют. Большинство из тех команд разработчиков, что мне довелось встретить, тратило на планирование и проектирование от силы несколько дней или недель (и это максимум), затем приблизительно от 6 до 18 месяцев отводилось на программирование и далее всего около двух-трех месяцев на отладку, тестирование и подготовку документации программы. Подозреваю, что нам нужно многому учиться у производителей кинофильмов. Затрачивай мы больше времени на фазу «предпроизводство – проектирование», мы сэкономили бы огромное количество времени программистов, которое стоит весьма дорого.

В киноиндустрии фаза предпроизводства является самой малозатратной стадией. Для создания подробной раскадровки дорогостоящей сцены погони со взрывами и спецэффектами требуется не так много денежных вложений. Если понадобится внести радикальные изменения, все что нужно – это стирательная резинка, карандаш и немного времени. Детальная прорисовка всех элементов на бумаге позволяет сэкономить миллионы на той стадии, когда включаются кинокамеры, а автомобили для съемки заполняются каскадерами и взрывчатыми веществами. Проходя этап подобной подготовки, вы инвестируете в свое время, которое позволит сэкономить бюджет и повышает возможные шансы на успех всего предприятия.

Если режиссеру вдруг вздумается показать взрыв вертолета вместо железной дороги, сделать это на этапе предпроизводства очень легко и незатратно. Внесение же таких изменений на этапе съемочного процесса смерти подобно. Создателям фильмов это известно слишком хорошо, поэтому они не спешат в период предпроизводства, пока все не продумают, а на этапе непосредственно съемки фильма строго следуют своему плану.

Почему же к разработке программных продуктов мы подходим совершенно иным образом? Мы посвящаем так мало времени и сил тщательному планированию, а вместо этого исписываем маркерные доски и создаем длинные электронные перечни с краткими названиями опций, а затем отправляем невероятно дорогостоящих программистов писать код для неведомого продукта. Равно как и представители сферы киноиндустрии, мы знаем, как дорого обходятся изменения на этапе создания кода, тем не менее это все еще не побуждает нас тратить больше времени и сил на планирование. Вместо этого мы нанимаем программистов, позволяем им приступить к написанию кода, а потом ссылаемся на то, что уже не можем вносить изменения, потому что затратный процесс создания кода теперь нельзя остановить.

Цикл доработки уже существующих продуктов претерпевает еще большие искажения. Если продукт уже вышел на рынок, для проектирования его ежегодной обновленной версии может в принципе *не быть* времени в графике. Вместо этого руководитель, ответственный за разработку продукта, просто составляет список опций на основе запросов потребителей и беззастенчиво передает его напрямую программистам для реализации, и так происходит каждый год.

Добавление в процесс разработки программного продукта этапа проектирования может так же, как и фаза предпроизводства в киноиндустрии, обеспечить невероятные преимущества. Детализированный план разработки программного продукта на бумаге позволяет избежать неопределенностей на этапе программирования, а также существенно снизить риски, традиционно связанные с процессом выпуска подобных продуктов.

Большая сделка

Руководство должно взять на себя обязательство осуществлять процесс проектирования до начала этапа программирования. Обращаясь к аналогиям, можно сказать, что проектирование взаимодействия представляет собой архитектурное планирование, а не дизайн интерьеров. Оно определяет, в какой каркас будет залит бетон для фундамента, равно как и из какого материала лучше пошить портьеры. Это обязательство должно давать проектировщикам моральное право единолично устанавливать форму и структуру программного продукта, а затем представлять их программистам. Конечно, это послужит серьезному сдвигу в культуре компании, однако программисты от этого станут лишь счастливее, а вы получите свои выгоды от значительного сокращения времени программирования и создания гораздо более совершенного продукта.

Желая обладать подобной властью, сообщество проектировщиков тоже должно взять на себя как минимум два обязательства. Первое – проектировщикам нужно иметь свой кровный интерес в этой игре. Им следует перестать оставаться на боковой линии, лишь давая программистам советы и безвольно предоставляя им полную ответственность за успех готового продукта. Только лишь обладать правильным ходом мыслей оказывается недостаточно. Нужно добиваться, чтобы указанные верные идеи реализовывались на практике, а такая возможность будет, только если проектировщики взаимодействия намеренно поставят себя под удар. Программисты делают это каждый раз, когда пишут очередную строку кода. Второе обязательство для проектировщиков – документировать свои предложения по проектированию.

Документируйте идею, чтобы она воплотилась в жизнь

За свою жизнь я усвоил один по-настоящему жестокий урок: никакое проектирование, даже самое лучшее, не будет иметь смысла, до тех пор пока не воплотится в жизнь. А этого не случится, пока оно не будет описано в мельчайших конкретных деталях, в концепциях, понятных программистам. Описывать идеи проектирования нужно в письменном виде, максимально подробно, приводя примеры и неопровержимые доказательства пользы. Этот документ должен быть напечатан и размножен многократно. Его следует собственнолично представить всем участникам команды, ответственным за продукт. На этом собрании непременно нужно

присутствовать вице-президенту по разработке, который в это время должен кивать и улыбаться. А лучше, чтобы это был сам президент компании.

Проектировщики должны писать, рисовать раскадровки, создавать анимации и прототипы своих решений максимально полно и с большим количеством деталей, чтобы программисты могли руководствоваться этими документами как планами и создавать код прямо по ним. В документе должно быть подробно описано достаточное количество примеров, чтобы вселить в программистов уверенность, что это решение действительно надежно и способно выдержать этап реализации.

Задокументированный план проектирования подобен плану военных действий. Каждый знает свою зону ответственности и осведомлен обо всех критических и важных аспектах. Теперь все будут действовать согласованно и сообща, создавая продукт под пользователя с конкретными потребностями.

Программисты обычно используют весьма действенную тактику, называемую «пассивно-агрессивной». Они не будут вступать в конфронтацию для решения какого-либо вопроса, а вместо этого начнут избегать привлекать внимание к этой проблеме и втихомолку сами будут действовать – или бездействовать, словно пассажир каноэ, который легонько наклоняется то влево, то вправо, тем самым задавая направление движения. Одна из моих самых любимых аксиом в бизнесе – это «Если чего-то не существует на бумаге, значит, этого не существует в принципе». В мире разработки программного обеспечения эта аксиома становится особенно справедлива. Если что-либо не было задокументировано, очень велики шансы, что идею истолкуют неверно или вовсе проигнорируют, поскольку внутренние мотивы программистов и пользователей различаются самым кардинальным образом. Просто *не описывать* диалоговое окно в документе нельзя – проектировщик обязан невероятно четко указать, где этого диалогового окна быть *не должно*, чтобы программисты не добавляли лишние окна по собственной инициативе. Программисты считают диалоговые окна отличной вещью и думают, что делают пользователям огромное одолжение, когда, улучив свободную минутку, вставляют в программу еще пару-тройку диалоговых окон. Для пользователей же эти диалоги – вещь крайне ненавистная, они изнуряют и снижают производительность работы пользователя.

Проектировщики взаимодействия, подобно архитекторам, подготавливают планы, которые описывают будущий продукт. Однако, несмотря на схожесть архитектурных планов и документации по разработке ПО, есть между ними и существенные различия. На архитектурных планах информация представлена в более сжатой форме. Одна линия на чертеже может подразумевать стену из 100 000 кирпичей. Когда же речь заходит о проектировании взаимодействия, сжатый формат подачи допускается лишь минимально. Описание поведения ста страниц кода может занять сто страниц в документации к программе. Я шучу лишь отчасти, когда говорю, что *тщательно описанная спецификация совершенно не отличима от кода, который ее реализует*. В идеальном мире разработки программного обеспечения проектировщикам бы даровали не менее года на подготовку плана продукта, а затем только три месяца дали бы программистам на его реализацию. В *реальном* же мире все ровно наоборот.

Все вышеописанное подразумевает, что в проектировочной документации по продукту должны быть опущены некоторые аспекты. Проектировщик должен обладать не только ощущением качества в отношении проектирования, но также уметь отличать действительно важные на текущий момент вещи. Он должен решить, какие части программного продукта подлежат тщательному проектированию, а какие можно оставить на волю программистов.

Я составляю проектировочную документацию, применяя методику «спирали», характерную для газет. В заголовке новости отражается вся важная информация о событии. В первом абзаце событие описывается снова, с чуть большим количеством подробностей. Следующие три абзаца повторяют важные моменты и сообщают дополнительную информацию. Оставшаяся часть текста может занимать несколько полос, в ней событие описывается с максимально возможной детализацией. Такая структура позволяет читателю узнать, что ему требуется, без погружения в излишние подробности.

Программный код зависит от проектирования

Существует расхожее заблуждение, что проектировщики взаимодействия создают дизайн пользовательского интерфейса. Нет сомнений, что дизайн интерфейса немаловажен, однако для процесса проектирования его роль не столь существенна, все равно как упаковка товара в розничной торговле. Дизайн пользовательского интерфейса – это та часть создания продукта, которая выполняется только после того, как будет полностью описано назначение и поведение продукта, обладающего интерактивностью. Упаковывая некачественный продукт в красивую и привлекательную обертку, вы по-прежнему имеете дело с некачественным продуктом.

К специалистам по дизайну интерфейсов обычно обращаются в тот момент, когда основная разработка продукта близится к концу или завершена. Возможностей для внесения значительных изменений в проектирование уже нет, и все усилия проектировщиков, вне зависимости от того, насколько они героические, мало на что могут повлиять.

Проектирование взаимодействия способно самым радикальным образом повлиять на процесс реализации продукта, однако это вовсе не значит, что оно касается только этого аспекта. Так, программисты обычно ожидают, что проектировщики обозначат, как будет выглядеть важное диалоговое окно внешне, со стороны пользователя. В то время как проектировщики могут захотеть поменять это диалоговое окно на какой-то другой компонент, например на панель инструментов. При этом их не интересует, как будут реализованы в коде диалоговые окна, равно как и панели инструментов.

В связи с тем что проектировщики взаимодействия оказывают существенное влияние на те аспекты, которые будут или не будут реализованы в конечном продукте, при этом старательно избегая описывать, как именно они будут реализованы, процесс проектирования можно считать процессом определения программного продукта. Фактически проектировщики определяют, что будет представлять собой внутреннее устройство продукта, путем описания его внешней структуры. Аспекты взаимодействия с пользователем описываются конкретно и точно, а решение вопросов реализации остается за программистами.

Польза проектировочной документации для программистов

В главе 3 «Пустые растраты» я задавал вопрос: «Как выглядит завершенный продукт?». Так вот, основной целью документации, которую составляет проектировщик взаимодействия, является получение ответа на этот вопрос. Если говорить в общем, то готовый документ с планами проектирования – это надежная система контроля процесса создания кода. Он выполняет ту же функцию, что сценарий и раскадровка на этапе производства фильма, то есть делает понятным для всех участников процесса, какими способами решать задачи, какие материалы потребуются подготовить и использовать, а также определяет, при каких условиях продукт будет считаться завершенным. Традиционно в процессе разработки наиболее сложен для контроля этап написания кода. Ему сопутствует больше рисков, нежели другим этапам, а потому неясность с тем, что считать «готовым продуктом», обходится в значительные суммы.

Понятный, детально прописанный документ позволяет руководителям компании получить более точное представление о том, каков будет конечный продукт, благодаря чему компания может сосредоточить усилия на конкретных направлениях. Имея на руках такое описание, руководители могут описывать все концепции в конкретных понятиях и тем самым более четко донести идею продукта до инвесторов, партнеров, сотрудников и коллег. Это дает гарантию того, что усилия всех участников процесса будут сосредоточены на достижении единой цели.

Программистам требуется разумный и сильный лидер. Ведь они и сами такие, а потому предпочитают идти за тем, кто похож на них, а не за тем, кто уступает им в способностях. Джерри Вайнберг утверждает, что всем известна такая истина: «Легче найти плохого руководителя, чем хорошего программиста». Так программисты сразу оказываются в более выгодном положении, невзирая на то что номинально руководители находятся выше в иерархической структуре компании.

Если руководители, ответственные за разработку продукта, неспособны четко и с убедительной ясностью описать, что именно должно получиться в результате работы их команды, они показывают свою слабость. Обычно желания руководства выражаются в виде необоснованных жестких дедлайнов и списков специфических опций, за которые идет торговля.

Как будет выглядеть конечный продукт, бывает доподлинно известно лишь программистам, а потому они обладают большим контролем над ситуацией, нежели их руководители.

Мне довелось работать с программистами, которые принимали в штыки нас как стороннюю компанию, взявшую на себя проектирование взаимодействия. Все, что им было известно о моей работе, – это что мы занимались «дизайном и проектированием», а еще они знали, что дизайн и проектирование – это самая творческая и самая увлекательная часть работы, которую обычно выполняли *они*. Тем не менее, поработав с нами некоторое время, они осознали, что мы не только не лишаем их работы, но и значительно улучшаем результат.

Не так давно я имел «удовольствие» присутствовать на одном совещании, проходившем в достаточно язвительном тоне, – на него пригласили программистов клиента, не обозначив им нашу роль в проекте. Речи одного седобородого программиста по имени Фред оказались особенно выдающимися. По ходу всего собрания, как только мы высказывали какую-либо идею или предлагали новый способ представления информации для пользователя, Фред начинал яростно нападать на нас. Он был невероятно умен и очень хорошо излагал свои мысли, а еще он был очень громогласным. Каждый раз, когда мы показывали очередной слайд, отображавший, насколько изменится пользовательское взаимодействие, он закатывал глаза и презрительно усмехался, а затем отпускал какой-нибудь комментарий, вроде того, что мы не осознаем, «какие героические усилия для этого придется приложить» ему и его команде. Он постоянно пытался показать, что все наши решения совсем не упрощают его работу, а лишь задают больше задач.

В конце концов, когда совещание завершилось и все стали расходиться, нашей команде удалось провести с Фредом личную беседу. Мы попытались объяснить ему, что нашей основной задачей является сделать программу легче в обращении и эффективнее в работе для *конечных пользователей* и что наши решения совершенно ясно приведут к необходимости дополнительных размышлений на этапе программирования. Мы твердили, что делаем все это во благо конечного пользователя. Неожиданно на лице Фреда отразилось крайнее удивление. Затем он воскликнул: «Это *серьезный вызов моим технологическим навыкам*!» Он моментально изменил свое отношение к ситуации, как только мы преподнесли ему «священный Грааль» всех программистов: сложную проблему, достойную его способностей.

Нашей целью не было испугать его – мы хотели дать ему то, чего он желал больше всего: шанса еще раз доказать всем, что он самый умный, находчивый и опытный программист во всей округе. Он увидел ситуацию под другим углом в тот момент, когда понял, что мы отнимаем у него только ту «грязную» и нелюбимую им сторону его работы, что связана с пользователями, оставляя ему в чистом виде только создание внутренней алгоритмической части программы. Из заклятых врагов мы превратились в его благодетелей.

Польза проектировочной документации для маркетологов

В большинстве компаний, чья деятельность не связана с компьютерами, за определение продукта отвечают профессиональные маркетологи. В индустрии разработки программного обеспечения маркетологи из этого процесса исключены. Все, с чем они имеют дело, – это обработка запросов на функции. Если они требуют от программистов каких-либо исправлений в работе программы, то в них попросту летит график сдачи проекта с жесткими дедлайнами и негодующий вопрос: «Как мы должны внести изменения, если у нас нет на это времени?» Руководитель по маркетингу не рискует нарушать драгоценные сроки, поскольку тогда команда не просто выбьется из графика, но и обнародует тот факт, что график – это просто фикция, после чего программисты будут безнаказанно злоупотреблять этим в будущем. Маркетологам известно, что на список опций не стоит полагаться, а потому они настаивают на большей вовлеченности в процесс определения продукта. К несчастью, маркетологи не обладают той степенью влияния, чтобы давать программистам указания, в которых те могли бы увидеть пользу или разумное зерно.

У хорошего процесса проектирования – а также у тщательного документирования, что тоже является частью этого процесса, – есть одно очень важное преимущество: несомненная польза, которая становится доступной и маркетологам. От маркетологов проектировщики узнают, какие невосполненные потребности или желания пользователей те пытаются удовлетворить. Проектировщики взаимодействия далее изучают указанных пользователей, чтобы выявить их

цели и сформировать набор персон. Предельно точно описанные персоны – неотъемлемая часть документации к проекту, эти образы пользователей впоследствии становятся основной точкой приложения маркетинговых усилий. Несмотря на то что программисты работают только со строками кода, персоны пользователей наполняют этот код смыслом. Несмотря на то что маркетологи работают только с каналами сбыта, целевыми сегментами рынка, средствами массовой информации и торговыми посредниками, персоны пользователей наполняют смыслом и все перечисленные аспекты тоже. Теперь программист и маркетолог могут достичь взаимопонимания.

Фактически проектировщики взаимодействия исполняют роль посредников между программистами и маркетологами, они способны переводить концепции с языка одних на язык других. Когда маркетологи затрудняются четко описать свои измышления, они могут обратиться к проектировщикам, и те помогут выразить задуманное в концепциях персон. А дальше проектировщики могут перевести полученную информацию в плоскость пользовательского взаимодействия и оформить в виде спецификации. Более того, теперь маркетологу будет понятно, кому и как адресованы его идеи, так что он может быть уверен, что ему не придется иметь дело с чисто технологической подоплекой продукта и пытаться найти на это покупателей.

Профессиональным маркетологам процесс разработки образов персон, как правило, очень хорошо знаком. Им часто приходится прибегать к подобному инструменту, составляя персон на основе характеристик покупателей продукта. Однако для этой задачи они используют исследования каналов сбыта и демографического состава вместо выявления целей и сценариев поведения, а потому их персоны обычно выглядят несколько иначе. Тем не менее эти персоны всегда бывают полезны маркетологам для создания их собственных планов. У маркетологов появляется возможность очень четко донести до *покупателя*, в чем выгода от использования продукта для *пользователей*.

Польза проектировочной документации для специалистов техподдержки

Каждому техническому писателю известно, что чем качественнее проектирование, тем меньшего объема документации оно требует. В отсутствие запутанного пользовательского взаимодействия отпадает необходимость в объемных и сложных объяснениях. У создателей документации появляется возможность посвятить сэкономленное время написанию текстов другого уровня. Теперь им не придется водить пользователя за руку по дебрям извилистого интерфейса – вместо этого они могут вложить свои старания в более высокий уровень воздействия на пользователей и помочь им в решении проблем предметной области, что гораздо более выигрышно. Например, не имея необходимости рассказывать о том, в каком месте инвентарной системы хранятся файлы, создатели документации могут поведать о том, как правильно эту инвентаризацию организовать.

То же самое справедливо и в отношении технической поддержки. Чем качественнее спроектирован продукт, тем меньше звонков будет поступать от пользователей. Как можно было увидеть из примера со сканером Peacock, речь о котором шла в предыдущей главе, задокументированное проектирование значительно снижает потребность в технической поддержке.

Польза проектировочной документации для руководителей

Самую значительную выгоду из всех получают руководители, ответственные за разработку продуктов. При качественном описании того, что должно получиться в результате, еще до этапа программирования, весь процесс разработки продукта протекает в разы быстрее, на всех этапах присутствует вся необходимая информация, риски снижаются, а затраты уменьшаются. Общая эффективность процесса становится выше, а сам процесс обретает самостоятельность – он не управляется необдуманными желаниями клиентов.

Проектирование, прежде всего, подразумевает большую предсказуемость. Это значит, что на этапе программирования будет более понятно, что может произойти и чего ожидать. Более предсказуемым становится и потенциальный успех продукта – его становится проще оценить. Два этих аспекта разработки программных продуктов считаются наиболее рискованными и

затратными. При таком подходе снижается стоимость этапа производства, а миф о непредсказуемом рынке удается преодолеть. Эд Форман – вице-президент, ответственный за разработку продукта в компании *Elemental Drumbeat*, – говорит: «Я измеряю выгоды от услуг проектировщиков в сэкономленных денежных средствах инвесторов».

Польза проектировочной документации для всей компании в целом

Для того чтобы построить успешный бизнес, нужно убедиться, что все участники процесса работают сообща для достижения единых целей. Любая неясность или неверная трактовка целей двукратно рассеивает все прилагаемые усилия. Во-первых, втуне пропадают старания тех, кто совершает действия в неверном направлении, а во-вторых, эта противодействующая сила тормозит тех, кто делает попытку следовать в нужном направлении. Так, если в каноэ один участник команды гребет в направлении, противоположном правильному, вся команда уже не может рассчитывать на призовое место. Чтобы прийти к победе, вся команда должна грести сообща, а любой человек, перетягивающий одеяло на себя, способен повредить всем.

Точно зная, что именно вы *делаете*, вы убережете себя от распыления усилий на вещи, которые точно делать *не собираетесь*. Мало какая компания может позволить себе тратить массу усилий на вещи, совершенно не соответствующие общим целям.

Уводя компанию от управления дедлайнами и споров по поводу списка опций, четко прописанная документация позволяет всем сосредоточиться на качестве продукта, которое вследствие этого неизбежно и резко улучшается. А это, в свою очередь, приумножает еще более ценный ресурс компании: лояльных клиентов.

Кто отвечает за качество?

Если ответственность за качество продукта распылена на всех участников процесса, значит, за это в конечном счете не отвечает никто. Всегда кажется, что проблемами качества занимается кто-то другой, например ваш коллега, пока вы работаете над другими задачами. Программисты несут ответственность за устранение всех неполадок в программе. Отдел продаж единолично отвечает за закрытие сделок, а отдел маркетинга – за упаковку продукта и его позиционирование. Так и получается, что при этом ответственных за качество и целесообразность продукта на данный момент нет. Иногда участникам недостает инструментов для определения и решения проблемы. Иногда они обнаруживают нехватку умений, которые бы позволили им корректно донести свое видение решения. А иногда для реализации решений им не хватает влияния в компании.

Как мы могли убедиться из предыдущих глав, загруженность программистов написанием кода вынуждает их поступаться необходимостью задумываться о целях пользователя. Руководителям, ответственным за разработку, тоже есть чем заняться, а потому им не до того, чтобы фокусироваться на нюансах поведения продукта. Маркетологи, с их недостаточной технической подкованностью, не имеют достаточных навыков, чтобы излагать свои мысли в терминах технических специалистов, что снижает уровень доверия к ним у программистов. Если же тщательно подготовленной документации по проекту нет, надежда на то, что продукт будет реализован эффективно и должным образом, ничтожно мала.

Основная мысль, которую я хочу донести посредством этой книги: *в конечном счете за качество продукта должны отвечать только проектировщики взаимодействия*. Им должна быть доступна возможность определять содержание программы и ее поведение. Именно проектировщики должны составлять списки опций и по большей части и график разработки тоже. Проектировщики – это адвокаты пользователя, а потому должны держать в своих руках контроль над внешними аспектами продукта.

В обмен на такую власть над продуктом проектировщики наделяются и рядом значительных обязательств. До тех пор пока у проектировщиков не будет ответственности, помимо власти, они не заслужат уважения у программистов, и последние быстро перехватят власть обратно. У проектировщиков должен быть кровный интерес в успехе всего проекта. В обязанности команды проектировщиков нужно вменить проектирование продукта, который возможно реализовать, который прост в использовании и обладает настолько высоким уровнем желанности, что пользователь может достигать своих практических целей, не попирая цели личные. Кроме того,

проектировщики должны создать настолько детализированные документы со спецификациями продукта, чтобы программисты были способны реализовать свою часть на основании этих предписаний. Проектировщики также должны обеспечить маркетологов понятными, задокументированными описаниями образов пользователей, а также того, каким образом продукт удовлетворит пользовательские потребности. Наконец, важнейшим является то, что проектировщики должны взять на себя ответственность за качество финального продукта.

Дружественный процесс проектирования

Из предыдущей главы было видно, как много профессионалов, поручавшихся за процесс проектирования, в итоге так и не добивались успеха в этом предприятии. Мы видели, как с этим пытались справиться тестировщики юзабилити, промышленные дизайнеры и другие специалисты, – все они потерпели поражение. В настоящий момент в индустрии нет сообщества, какого бы то ни было размера, способного справиться с этой проблемой.

Ряды проектировщиков взаимодействия пусть медленно, но все же пополняются, и здесь нужно четко отдавать себе отчет в том, что самое важное – это сделать процесс разработки дружественным к процессу проектирования, а не собрать команду из самых талантливых проектировщиков. Очень важно прийти к соглашению, что на проектирование будет выделено время и что это время будет предшествовать этапу создания кода. Самый выдающийся проектировщик взаимодействия в мире окажется бессилен, если бета-версия продукта должна выйти уже на следующей неделе.

К примеру, многие из тех компаний-разработчиков программного обеспечения, что сейчас представляют собой могущественные корпорации с долгой историей, взросли пóтом и кровью очень юных и весьма неопытных программистов. Вполне вероятно, что этим программистам была предоставлена невероятная свобода действий, а подобный союз огромной ответственности и внушительных полномочий нередко является тем самым очагом, где зарождается пламя величия. То же справедливо и в отношении проектирования взаимодействия. Если человек оказывается наделен такой ответственностью за качество продукта, а также соответствующими полномочиями, он с большой долей вероятности примет этот вызов, вне зависимости от своего уровня подготовки. Так, если вы разыщете подходящего специалиста и вмените ему полную власть над качеством и поведением продукта, вы получите продукт много лучший, чем если бы вы этого не сделали. Суть проблемы кроется в процессе, а не в людях. Конечно, при прочих равных условиях лучше давать задачу профессионалу с соответствующим уровнем подготовки. Тем не менее, если такого специалиста поблизости нет или оплата его услуг не заложена в бюджет проекта, все равно лучше работать с менее опытными практикующими проектировщиками, чем просто отдать все на откуп программистам.

Кого в данном случае можно считать наиболее подходящим специалистом на роль проектировщика? Наибольшая польза будет от человека, который отдален от процесса непосредственного конструирования продукта, а потому может беспристрастно поставить себя на место пользователя. Программист тоже может быть таким человеком, но только если он не занят в создании этой программы, – в противном случае возникнет конфликт интересов.

Откуда появляются проектировщики взаимодействия

Итак, вам по-прежнему нужно найти кого-то для выполнения проектирования взаимодействия. Стоит вам начать поиски, как вы увидите, что почти в каждой высокотехнологичной компании такие проектировщики есть и они преимущественно разочарованные: это технические писатели, к которым обращаются программисты с просьбой помочь прояснить некоторые вещи; менеджеры по продукту, книжные полки в кабинетах которых ломятся от книг по проектированию взаимодействия; юзабилити-тестировщики, упоминающие о том, что в процесс разработки следует вовлекаться на ранних стадиях; маркетологи, акцентирующие внимание на том, что им удалось приобрести стереосистему с наименьшим количеством кнопок; программисты, которые пишут совсем незначительное количество строк кода, но при этом другие программисты просят тех поработать с ними. На самом деле, когда сотрудникам компании становится известно, что проект начнется с этапа проектирования и случится это до этапа программирования, кто-то из них непременно вызовется,

утверждая, что *горит желанием* быть именно тем человеком, который отвечает за качество продукта.

Когда вы станете подыскивать кандидата на полный рабочий день, хороший специалист может и не назваться проектировщиком взаимодействия. Вам следует обращать внимание на людей, в общих чертах понимающих технические детали, но обладающих страстью к дизайну и проектированию. Таких людей можно обнаружить в совершенно различных сферах деятельности и с совершенно разным опытом за плечами. Когда я нанимал людей для своей студии дизайна и проектирования, я просил их выполнить тестовое задание, связанное с решением проектировочной задачи, так как понимал, что в их резюме может не оказаться нужных сведений. Сейчас в моей студии в должности проектировщиков работают несколько человек, которые прежде были техническими писателями, руководителями разработки продуктов, специалистами технической поддержки или графическими дизайнерами. Многие из проектировщиков, работающих у меня, пришли из гуманитарной сферы, но есть и те, кто специализировался в физике, архитектуре, компьютерных дисциплинах и промышленном дизайне.

Специалисты с опытом работы в технической поддержке или в документировании спецификаций перспективны в отношении умения понять типичные потребности пользователя. Руководители, ответственные за разработку программного продукта, разбираются лишь в потребностях программистов, возникающих в процессе разработки. У графических и промышленных дизайнеров есть стремление к красоте и умение претворить ее в жизнь. Дизайнеры с гуманитарным образованием, имеющие опыт работы в высокотехнологичных отраслях, сочетают в себе понимание технологий и способность донести свою мысль.

Формирование команд проектировщиков

Дискуссия о том, как разные специалисты подходят к созданию команд проектировщиков и управлению ими, могла бы занять отдельную книгу. Здесь же я касаюсь некоторых методов проектирования, чтобы вы лучше понимали, что я имею в виду, употребляя термин «проектирование», однако даже не делаю попытки изложить всю методологию целиком. Тем не менее, благодаря своему опыту управления командами проектировщиков в моей студии, я могу предложить вам некоторые ключевые принципы.

Формируйте команды небольшого размера. В целях преуспеть проектировщики должны обладать единым видением результата. В моих командах обычно бывает от двух до трех проектировщиков на каждый продукт, которым периодически могут оказывать помощь и содействие другие специалисты. Если проект сложный, проектирование может доходить до той стадии, когда у продукта возникает несколько разных интерфейсов – тогда задачу можно разделить на части и каждую из них отдать на откуп собственной команде. Однако до этого момента ситуация будет как в известной поговорке: «У семи нянек дитя без глазу».

Команду следует изолировать от руководства и программистов. На начальной стадии проекта проектировщики обязательно должны провести беседы с другими специалистами, задействованными в процессе, чтобы составить четкое представление о проблеме и определить персон. После этого им нужно дать пространство для проработки самых тупиковых вариантов, с тем чтобы они выработали собственное наиболее удачное решение, но сделать это эффективно они смогут только в уединении.

Назначьте специалиста, ответственного за составление документации по ходу работы команды. Каждый из членов команды внесет в эту документацию свой вклад, однако отвечать за нее должен лишь кто-то один – только тогда можно добиться эффективной работы.

Предоставьте команде время, чтобы привести мысли в порядок. Позднее, когда основные задачи решатся, будет вполне разумно, если вы обратитесь к команде проектировщиков за выяснением специфических деталей. На ранней стадии проекта команда должна самым тщательным образом обдумать все аспекты решения проблемы и представить свои измышления в виде целостной концепции. Когда моя студия предоставляет клиенту полную структуру продукта, мы формируем документацию и демонстрируем свои решения относительно проектирования с неофициальными контрольными точками там, где это требуется. В ходе первой презентации мы очерчиваем проблему, представляем набор персон и обговариваем

задачи, которые требуется решить при помощи проектирования. На каждой последующей презентации мы описываем проект продукта во все более и более подробных деталях.

Организуя такой управляемый процесс, центром которого является проектирование, а не программирование, компании получают возможность избежать опасной игры с огнем. Им становится заранее известно, что требуется пользователям и каким образом им это предоставить. Им также станет известно, при каких условиях процесс разработки можно будет считать завершенным, а у специалистов различных направлений сформируется как общее, так и частное видение, которое позволит им сплотиться вокруг продукта.

14. Удовольствие и мощь

Для того чтобы в полной мере испытать весь эффект от проектирования взаимодействия, вам нужно сделать его одним из неотъемлемых компонентов процесса разработки программного обеспечения. Нельзя прибегнуть к нему только впоследствии.

В предыдущей главе я упоминал, что требования к дизайну и проектированию нужно документировать и делать это до начала процесса создания кода. Тем не менее в водовороте задач по разработке продукта у программиста все еще остается возможность просто-напросто проигнорировать документ проекта, вне зависимости от детализации его исполнения. Такая ситуация весьма характерна для пассивно-агрессивной культуры поведения разработчиков программ, при которой инженерам свойственно относиться к любым указаниям по проектированию исключительно как к рекомендациям, которые можно учитывать лишь при наличии возможности и свободного времени.

До всех участников проекта следует предельно ясно донести мысль, что предложения по дизайну и проектированию – это руководство к действию, а не просто праздные рассуждения. До тех пор пока в компании открыто и массово выражается неприятие процесса дизайна и проектирования, разработчикам будет казаться, что только от них зависит успешность создаваемого продукта.

Существует только один способ донести эту мысль наиболее эффективным образом. Топ-менеджмент компании должен ясно дать понять всем менеджерам, ответственным за проектирование и разработку, что отныне программисты в этом участия не принимают. Ко всем должно прийти осознание, что качество продукта теперь в компетенции команды проектировщиков, которая, в свою очередь, с этого момента имеет право указывать, что делать, разумеется, с разрешения руководства.

Программистам позволено импровизировать внутри программы, а любой выявленный аспект пользовательского взаимодействия должен быть реализован в соответствии с предписаниями. Никто не говорит, что предписания нельзя оспорить или усомниться в них, но изменять их в одностороннем порядке или пренебрегать ими недопустимо. Не следует относиться к предписаниям специалистов по дизайну и проектированию как к необязательной рекомендации, которой можно следовать лишь частично или изменить на свой лад.

Команда проектировщиков отныне несет ответственность за каждый элемент, с которым взаимодействует пользователь. При этом сюда относится не только программное, но и аппаратное обеспечение. К этому следует также отнести и сопутствующее программное обеспечение, такое как установщики программ и модули технической поддержки.

Это требование в целях создания успешного продукта, пожалуй, является наиболее радикальным, следование ему повлечет определенную перестройку в культуре компании, к которой нужно будет привыкнуть. Позже в этой главе мы уделим более пристальное внимание вопросам культуры. А сейчас проанализируем опыт компании, которой удалось успешно внедрить проектирование в процесс разработки.

Пример успешного внедрения проектирования

Моя студия дизайна и проектирования не так давно закончила работать над одним из наших самых успешных проектов для небольшой компании в регионе Pacific Northwest. Компания называется *Shared Healthcare Systems, Inc.* (*SHS*), она занимается созданием программного обеспечения для управления всеми аспектами предоставления долговременных услуг здравоохранения.

Во время первых встреч с заказчиком я прилагал массу усилий, чтобы объяснить важность персон в проектировании и рассказать о том, как мы используем их в этом процессе. К нашему

великому удивлению и удовольствию, команда *SHS* всерьез прониклась этой идеей. На первом организационном совещании по проекту они показали ранее подготовленный ими набор из порядка десяти персон и их характеристик. Конечно, нам все равно потребовалось уделить внимание изучению предметной области, чтобы удостовериться в корректном подборе персон и проработать детали, но нам уже не нужно было убеждать разработчиков и маркетологов в необходимости применять этот инструмент.

Бизнес *SHS* вовлек компанию в ситуацию, которую Мишель Борк из компании *Clinidata* в Монреале характеризует как «клинический ураган». Процессу компьютеризации малого бизнеса первыми подверглись кабинеты врачей, но затронута оказалась только сторона работы, связанная с оплатой услуг. Тот участок работы, на котором врач взаимодействует с пациентом, непреклонно противился приходу цифровой эпохи и оставался одним из последних бастионов некомпьютеризированной части мира.

И хотя большинство усилий *SHS* было направлено на решение административных задач, существенная доля работы велась прямо в эпицентре этого урагана. Ранее мы имели опыт составления планов по дизайну и проектированию для других компаний в этой сфере, но мы никогда не были в самой гуще событий. Эта задача стала для нас серьезным вызовом и невероятно вдохновляла.

Специалисты компании *SHS* тоже воодушевились и поначалу заявили нам, что их бизнес настолько масштабен, что они не слишком уверены в нашей способности совладать с такими масштабами. В *SHS* полагали, что их бизнес элементарно слишком велик и сложен для понимания. Для нас это звучало как вызов, и мы с готовностью взялись за дело.

Проект был действительно *грандиозным*. Мы определили *пять* ключевых персон, что было почти в два раза больше, чем в любом из наших предыдущих проектов. Сначала такое количество показалось нам подозрительным, но в ходе дальнейшего анализа мы обнаружили, что SHS действительно пытается охватить широкий сегмент рынка здравоохранения. Очевидно, что слишком трудно за раз создать программное обеспечение с учетом потребностей всех пяти ключевых персон. Компания SHS осознала этот факт, после чего разработку и дизайн продукта разбили на несколько этапов – по одной персоне на каждый этап.

Нашим контактным лицом в *SHS* был Дэвид Вест – вице-президент по разработке, который, при прочих своих достоинствах, пользовался уважением и доверием других сотрудников этой развивающейся компании. Маркетологи, равно как и программисты, знали, что он действует исключительно в их интересах. Знали они и то, что он строг, но при этом справедлив. Он словно незыблемая скала в бурных волнах разработки. Его открытая приверженность процессу проектирования привела к тому, что другие разработчики стали больше доверять нашей работе и относиться к ней серьезно, как к спецификации.

Когда *SHS* впервые обратилась в компанию *Cooperг Interaction Design*, работа отдела разработки программного обеспечения была организована в соответствии с доставшимся ей программным продуктом, разделенным на два модуля – клинический и финансовый.

Когда мы закончили наше исследование и разработали модели персон, мы довольно быстро осознали, что текущая система не способна в полной мере удовлетворить запросы медицинских работников. Даже без учета серьезных проблем взаимодействия такое разделение программы на две части (клиническую и финансовую), каждая из которых относилась к своей информационной подсистеме, не имело под собой веских оснований. Оно лишь влекло за собой дополнительную бумажную работу, которую нужно было делать, потому что система не позволяла поступать иначе. Каждый пользователь оказывался изолирован на своем участке данных, так как между этими двумя системными модулями отсутствовала какая-либо связь.

Нашей рекомендацией было создать единую запись о пациенте, содержащую одновременно и клиническую, и финансовую информацию в одной консолидированной базе данных, а поверх этого – модульный пользовательский интерфейс, который бы позволил каждой персоне получить доступ только к той информации, которая нужна для решения конкретных пользовательских задач. Результатом стала реструктурированная база данных, лежащая в основе программы. Более же примечательным стало то, что *SHS* произвела реорганизацию отдела разработки в соответствии с новой архитектурой программы! Разработчиков распределили на две новые группы: одни работали с архитектурой записи о пациенте и базой данных, другие – с

интерфейсами для персон. В программные спецификации и документацию *SHS* по этому проекту теперь включали имена персон для более ясного определения их функционала.

Программисты *SHS* предприняли довольно мудрый шаг, отодвинув задачи по программированию на конечный этап процесса проектирования. Команда *SHS* во главе с Дэвидом ясно осознавала тот факт, что, хотя простой программистов может обойтись недешево, намного дороже будет «намертво» заполнить все лишним программным кодом.

Программисты занимались бэкендом программного обеспечения, не затрагивающим пользовательского интерфейса. Помимо этого, они разделили проект на несколько фаз, в ходе одной из которых был создан краткосрочный подпроект, позволяющий обеспечить функционирование существующего продукта на более высоком уровне. Таким образом, программисты были обеспечены задачами, при этом не нарушая хода долгосрочного стратегического проекта.

Чтобы оптимальным образом согласовать наши задачи с задачами программистов, мы разбили процесс на несколько крупных этапов.

Мы достигли договоренности использовать только двух из пяти персон на начальном этапе проектирования, а остальных трех задействовать позже. И вновь таким образом мы могли заниматься дизайном и проектированием еще до завершения процесса разработки, при этом программисты также не простаивали.

Понимание необходимости дизайна и проектирования на уровне компании

В большинстве компаний компетенции, связанные с проектированием основного продукта или услуги, считаются ключевыми. В мире высокотехнологичных программных продуктов и устройств существует тенденция – и весьма ошибочная – относить проектирование взаимодействия к ответственности инженеров. На самом же деле процесс создания продукта включает два этапа: проектирование и программирование. Готовность к тому, что проектировщики наряду с инженерами также должны принимать участие в работе над продуктом, составляющим суть бизнеса, требует ряда перемен в культуре компании.

Сотрудники любой компании, независимо от сферы ее деятельности, понимают, что за ними закреплены определенные обязанности. Например, в компании, изготовляющей катушечные обмотки для динамиков, управляющему производством известно, что одной из его задач является покупка лучшего сырья по самой низкой цене, но при этом он не имеет права подписывать контракт с поставщиком, пока совет компании не одобрит это решение. Управляющий производством не слишком осведомлен в юридических вопросах заключения контрактов, тем не менее он осознает, что не стоит ставить компанию под угрозу, пренебрегая советом профессиональных юристов по контрактам и сделкам. Не будучи специалистом в контрактном деле – а вернее, именно по этой причине, – управляющий знает, что необходимо обратиться за консультацией к юристу.

Человек, ответственный за приемку грузов в доке, даже будучи самым младшим сотрудником в штате, знает, что в его компетенции расписываться только за те грузы, доставка которых запланирована, и ни за какие другие.

Основатель и президент компании, изготовляющей катушки, тоже признает необходимость привлечения юристов на всех уровнях. У него нет специализированного юридического образования, поэтому он, прежде чем подписывать какие-либо документы, обращается за советом к помощнику.

И хотя никто из них не обладает должным уровнем знаний в области права, каждому из них более чем понятна необходимость проведения юридической экспертизы. Ни один специалист компании не возьмет на себя какие-либо обязательства до тех пор, пока юристы не вынесут свой вердикт. В компании присутствует осознанное понимание необходимости такого правового надзора и вмешательства профессиональных юристов, когда того требует ситуация.

Такая осознанность характерна и для других областей деятельности компании.

Когда компании по изготовлению катушечных обмоток потребовался новый производственный цех, она прибегла к помощи внешнего специалиста – архитектора. Несмотря на то что управляющий производством и президент компании оба обладали высоким уровнем квалификации в отношении организации производственных процессов, они понимали, что не

разбираются на должном уровне в нюансах строительства зданий и проектирования конвейеров. Никому бы и в голову не пришло расширять производственные площади, прежде не обратившись за советом к архитектору. Роль архитектора состоит в том, чтобы переводить запросы заказчика в концепции, понятные строителям.

То же самое будет справедливо и в отношении рекламы. Менеджер по маркетингу даже не подумает попросить заготовщика катушек описать преимущества продукта для рекламной брошюры или отраслевого журнала, посвященного акустическим системам. Каждый сотрудник компании, вне зависимости от имеющегося опыта, ясно осознает, что рекламой должны заниматься профессионалы и что только они могут сформировать публичный имидж компании на достойном уровне. Разумеется, этим могут заниматься как собственные сотрудники компании, так и внешние привлеченные специалисты рекламного агентства. Оба варианта заслуживают внимания.

Эта аналогия не в полной мере отражает суть, поскольку ни архитектурно-строительная, ни юридическая экспертиза не являются ключевыми компетенциями для производственной компании. Иначе обстоит дело с программированием. Программирование – это создание программного продукта, а именно это обычно и является ключевой компетенцией компании-разработчика. С учетом того факта, что продукт непосредственно влияет на успешность бизнеса, можно ожидать от любой компании осторожного отношения к тому, чтобы ненароком не отдать бразды правления в ненадежные руки, – даже более осторожного, нежели в применении к рекламе, архитектуре и закупкам.

Нам следует простроить в компании твердую осознанность в отношении того, что проектирование взаимодействия – это та сфера, где требуется участие профессионалов, и что интерактивный продукт нельзя отдать лишь на откуп инженерам, необходимо присутствие проектировщика, чтобы продукт стал успешным на открытом рынке.

Польза перемен

В мире, где преобладает программное обеспечение, апологеты так многочисленны и так влиятельны, что власть их ослабнет еще не скоро. Однако это непременно произойдет. И послужить тому причиной может осознание каждым человеком простой мысли: технологиям не обязательно быть столь бесчеловечными. Все чаще сталкиваясь с программным обеспечением, которое не будет уничижать их, они станут все более нетерпеливыми в работе с программами, которые только и делают, что изматывают их и приводят в замешательство. Громким топаньем они погонят медведей-плясунов с глаз долой.

В те времена, когда пользователи таких продуктов были немногочисленны, они сами принимали непосредственное участие в процессе, в связи с чем осознавали всю революционность технологий. Но чем шире стали распространяться технологии в повседневной жизни, тем меньшее количество причастных осознают, насколько эти технологические достижения велики. Они уже не готовы оказывать снисхождение некачественно спроектированным продуктам только потому, что процесс их создания был сложным и тяжелым.

Следовать веяниям новых технологий представляется неплохой идеей, только это обычно приводит к появлению скучных программных продуктов, которые являются лишь усложненными версиями своих предшественников. Проектирование взаимодействия позволяет разрушить этот шаблон и явить миру такие продукты, которые обладают невиданными до сих пор возможностями.

Продукт с качественным проектированием взаимодействия становится не только желанным для потребителей, но и получает в награду особое рыночное преимущество – лояльность покупателей. Если продукт сделал потребителя счастливым, то человек станет приверженцем вашего бренда и вашей компании на долгие годы. Но если вы выпустили на свет очередного медведя-плясуна, потребители станут озираться по сторонам, пытаясь найти более простые в использовании и более дружественные пользователю варианты.

Проектирование взаимодействия позволяет сократить временные затраты на разработку продукта. Заранее зная, какой продукт нужен потребителю, вы потратите меньше времени на поиск подходящего функционала методом случайного подбора.

Правильный процесс создания продукта в каком-то смысле всегда является итеративным. Для того чтобы верно сопоставить все детали, обычно требуется несколько попыток. За счет

включения проектирования взаимодействия в начальные стадии процесса можно добиться значительного уменьшения общего количества итераций. Выпуск каждой следующей версии продукта выливается в огромные затраты, а потому с уменьшением количества таких версий, скажем, с четырех до двух, вы получаете значительную экономию времени и денежных средств.

Снизив количество выпускаемых версий продукта и уменьшив количество производимого кода, вы удешевите процесс разработки. Нередко можно услышать от программистов жалобы по поводу того, что дизайн и проектирование требуют более сложного кода, и обычно так оно и есть. Однако если смотреть на ситуацию в целом, то кода в итоге все же получается меньше. Увеличение сложности кода не слишком сильно влияет на его стоимость, а вот в случае с увеличением его *объема* ситуация прямо противоположная. Для каждой лишней строки кода требуется дополнительное тестирование, отладка и поддержка.

Если у них нет хлеба, пусть едят пирожные

Местом моего проживания и работы является Кремниевая долина, штат Калифорния. Практически все, кого я знаю здесь, задействованы в индустрии высоких технологий. Все мы обеспеченны, высокообразованны, обладаем географической и социальной мобильностью, и еще мы прекрасно владеем компьютерами, мобильными телефонами, DVD-плеерами, банкоматами и любыми другими программными продуктами и устройствами на их основе, широко представленными на пестром рынке для среднего класса. За ланчем в Crescent Park Grill или Spago я слышу разговоры посетителей за соседними столиками, в которых они непременно обсуждают что-нибудь с клиент-серверной архитектурой или нечто веб-ориентированное. Это чудесное место для жизни, но очень оторванное от того, чем живет большинство людей в этой стране, если не сказать, мире. Здесь, в Долине, наши представления о пригодности высокотехнологичных продуктов легко подвергаются искажению. Мы забываем, как в действительности сложны наши продукты для использования.

Десять лет назад Сеймур Меррин – консультант по розничным продажам – как-то сказал, что нам проще убедить потребителей в том, что программы удобны в использовании, чем действительно сделать программы таковыми. Слова Меррина звучали довольно цинично, но одновременно он выражал удивление тем фактом, что эта откровенная ложь просто сошла нам с рук. В настоящее время его слова столь же актуальны, как и тогда, но с развитием высоких технологий мы не можем прикрываться одним лишь цинизмом – мы должны найти настоящее решение.

Люди осведомлены о том, что пользоваться компьютерами непросто, но полагают, что на это есть веские причины. Большинство из них думает, что все работает настолько хорошо, насколько это в принципе возможно.

Несмотря на то что обычные пользователи за пределами компьютерной индустрии крайне недовольны сложными в применении программными продуктами, другие люди, которые эти продукты создают, в целом довольны сложившейся ситуацией. Программисты не считают, что пользоваться компьютерами сложно, поэтому они готовы терпеливо выносить некоторые вещи, пока исследуют технологию и наслаждаются созданием новых крутых программ, похожих на медведей-плясунов.

Что касается остальных, они получают то программное обеспечение, которое им требуется, – а требовалось им до нынешнего момента очень мало. Разработчики программ и устройств предлагают нам множество «наворотов», излишних функций и «примочек», в которых нет необходимости и которыми мы едва ли воспользуемся, но все равно мы эти устройства покупаем. Нам нужно, чтобы программы работали стабильно, оттого разработчики подвергают их тщательному тестированию; в результате программы довольно надежны. Мы нуждаемся в обновленных версиях так быстро, как только это возможно, оттого разработчики выпускают их с невероятной скоростью. Но мы не требуем, чтобы программы были приятными и мощными, поскольку просто не осознаем, что это возможно, вот почему они остаются столь неэффективными и удручающими.

Пользователи продолжают питать смутные надежды, что приход следующей технологии, например связанной с распознаванием речи, сделает программные продукты более удобными и простыми в использовании. Думать таким образом несколько наивно и не слишком разумно. Мне жаль наблюдать, как апологеты жестоко культивируют подобные надежды.

Программное обеспечение похоже на пластичную массу – из него можно смоделировать все, что пожелает создатель. Разработчики не делают программы простыми вовсе не потому, что это невозможно, а потому, что они не умеют этого делать. Отказываясь признаваться в таком постыдном факте, они утверждают, что нельзя сделать более удобный продукт из-за «технических ограничений». Пользователям компьютеров, далеким от программирования, приходится соглашаться с мнением разработчиков и мучиться – или не соглашаться и (что бы вы думали!) все равно мучиться. Не имея экспертных знаний в области разработки, они не могут предложить собственное решение проблемы, так что их считают просто непродуктивными нытиками.

Автомобильные заводы штата Детройт производили огромных хромированных чудовищ, потребляющих невероятное количество топлива, и при этом надменно заявляли: «Мы даем нашим клиентам именно то, чего они хотят». В период волны нефтяного кризиса, накрывшей Америку в 1970-е годы, японцы вывели на рынок бюджетные автомобили небольшого размера, чем нанесли Детройту сильнейший удар, который навсегда останется в истории этого штата. В настоящее время индустрия производства автомобилей в Америке ценит желания потребителей гораздо выше и уже не рискует самонадеянно утверждать, что им все известно лучше.

Японские производители захватили значительную долю рынка автомобилей благодаря тому, что предложили потребителям нечто, чего те никогда и не осмеливались желать. Однако они сразу смогли выявить качество, когда увидели его своими глазами. Аналогично большая доля рынка проектирования программного взаимодействия остается незахваченной, компании сражаются за то, чтобы занять это теплое место. *Microsoft* сегодня не менее уязвима, чем была компания *General Motors* в 1974 году.

Массовый рынок для потребителей, не слишком хорошо управляющихся с технологиями, быстро перейдет на простые в применении продукты, как показывает практика ошеломительного роста популярности веб-среды. Те же люди, которых привлекла легкость взаимодействия с интернетом, аналогичным образом быстро привыкнут к качественно спроектированным программным продуктам, которые позволяют с легкостью решать сложные задачи.

Те мало знакомые с технологиями пользователи, обитающие за пределами анклавов вроде Кремниевой долины, не способны добиться каких-либо изменений просто потому, что не обладают сплоченным единством. Конечно, они могут отличить хорошее от плохого, но только тогда, когда эти продукты уже созданы и продаются в магазинах.

Каких-либо изменений можно ожидать лишь в том случае, когда люди, которые непосредственно задействованы в процессе создания продукта и способны оказать влияние на ранние этапы его разработки, будут заинтересованы в решении проблемы. Так как у программистов наблюдается конфликт интересов, я направляю свой призыв к апологетам, находящимся в самом сердце индустрии высоких технологий. А если человек сегодня занимается бизнесом, он оказывается вовлеченным в эту индустрию, хочет он того или нет. Едва ли в мире остались компании, которые не участвуют в процессе приобщения к новым технологиям или уже не ставшие зависимыми от них.

Ни один из существующих ныне современных программных продуктов не способен подарить удовольствие и мощь людям, не относящимся к помешанному на технологиях меньшинству. Сообщество программных инженеров просто-напросто утверждает, что людям нужно становиться «компьютерно грамотными». Думаю, что со временем эту мысль будут трактовать так же, как известное снисходительное высказывание Марии-Антуанетты: «Если у них нет хлеба, пусть едят пирожные!» Французская революция даровала людям хлеб, а грядущая революция в проектировании одарит людей технологией.

Как изменить процесс

Многие из тех разработчиков и технических руководителей, что в данный момент создают программные продукты, поступают так, как поступают, потому что верят в правильность такого подхода, но при этом не считают его нерушимым. Они достаточно прагматично смотрят на мир, чтобы внедрить изменения, стоит им увидеть эффект от проектирования взаимодействия. Если им донести всю ценность дизайна и проектирования, то, по моему опыту, они загораются желанием включить его в процесс разработки программного обеспечения.

За долгую историю своего существования разработчики программного обеспечения не раз меняли свои убеждения. Разумеется, они инженеры и всегда будут мыслить как инженеры, но вместе с тем они склонны заимствовать новые подходы, даже самые радикальные, если будет явно доказана эффективность таких подходов.

Двадцать лет назад для инженеров из сферы программирования было нормой самостоятельно производить тестирование собственного кода. Фактически программисты справедливо полагали, что лишь они сами могут достоверно протестировать свой код, потому что только им лучше всех прочих известны все его слабые места и потайные уголки, которые нужно исследовать.

Поражает также то, что была справедлива и другая истина – несмотря на *необходимость* тестировать код самостоятельно, программисты почти повсеместно ненавидели это занятие и жалели потраченное время и усилия. Тем не менее они продолжали выполнять эту работу, так как верили, что их роль в этом процессе важна, а агрессивное тестирование эффективно.

Медленными темпами, в течение двух десятилетий к специалистам индустрии приходило осознание, что эту часть программистской работы может взять на себя группа профессиональных тестировщиков, и в итоге такая практика снимать с программистов ответственность за тестирование все же укоренилась в этой сфере. Когда схлынула волна первичного скептицизма, программисты поняли всю ценность этой идеи. Тестировщики же, к великому удивлению большей части программистов, в самом деле *получали удовольствие* от тестирования. Им нравилось придумывать новые и новые, все более нещадные методы тестирования программных продуктов, обнаруживать слабые места и уязвимости, вызывать исключения и доводить программу до самых невероятных случаев. Нет сомнений, что профессиональные тестировщики справляются с испытаниями кода намного лучше программистов. Программисты же, в свою очередь, поняли, что не только сбросили с себя значительную и не самую приятную часть работы, но и передали ее в руки тех, кто может выполнять ее более надежно, быстро, организованно и вдумчиво. В современной доктрине разработки программного обеспечения сказано, что оптимальным соотношением программистов и тестировщиков является один к одному. В настоящее время уже ни один программист не скажет, что он лучший тестировщик собственного кода.

Подобный плавный переход можно наблюдать, когда дизайн и проектирование становятся неотъемлемой частью процесса разработки программ. Самые значительные выгоды будут у тех, кто первым начнет применять такой подход к процессу.

Разработчики программного обеспечения и проектировщики взаимодействия преследуют одни и те же цели: обе категории хотят, чтобы их продукт стал успешным. Только инструменты и мерила этой успешности кардинально различаются у обеих групп.

Не имея веских доказательств, программист всегда начнет исходить из собственного опыта, интуиции, уровня подготовки. Интуиция убеждает его в том, что нужно еще больше функций. Опыт твердит, что нельзя допускать дилетантов, которые оперируют лишь домыслами и своими прихотями, к чувствительному, сложному и деликатному процессу разработки. Уровень подготовки говорит, что подходить к проектированию интерфейсов нужно, основываясь на собственных представлениях.

Проектировщик взаимодействия не может напрямую критиковать эти мотивы. Разработчики – люди слишком рациональные, чтобы так просто поступиться своим опытом в угоду мнениям других. Задача проектировщика – обозначить проблему под другим углом, а кроме того, доказать еще две вещи: что такое видение проблемы более эффективно и что оно не идет вразрез с уже имеющимися точками зрения.

Вне зависимости от того, насколько тверды убеждения проектировщика взаимодействий, шанс, что он разбирается во внутреннем устройстве программного обеспечения лучше, чем программист, весьма невелик. Другими словами, он не сможет увидеть проблему с такого ракурса, как видит ее программист. Для того чтобы проектировщик преуспел в своей деятельности, его точка зрения должна отличаться.

Точкой пересечения взглядов программистов и проектировщиков является точная и полная проектировочная спецификация. Когда проектировщики предлагают такие решения, которые кажутся программистам совершенно справедливыми, тогда к их мнению возникает доверие и на них становится возможным положиться.

* * *

В 1998 году в еженедельном издании *Business Week*[40] была напечатана статья, в которой колумнист Стивен Вилдстром затронул тему недовольных пользователей компьютеров. Отклики читателей выражали на удивление полярные мнения, их было так много и они отличались таким накалом страстей, что Стивен Вилдстром резюмировал ситуацию следующим образом:

Компьютерная индустрия характеризуется наличием большого количества разочарованных, сбитых с толку, несчастливых пользователей. Для благополучия сектора высоких технологий в долгосрочной перспективе это представляет более значительную угрозу, нежели азиатский кризис, проблема 2000 года или большинство угроз иного характера.

В статье были также приведены знакомые многим «крики души» потерпевших: «Из-за этого компьютера мне кажется, что я идиот!» – и не менее знакомые восклицания апологетов: «Пользователи сами не знают, чего хотят, а даже если знают, то все хотят разного. Они просто невнимательно читали руководства и плохо изучили программу!» В заключение Стивен приводит такое интересное наблюдение:

Среди всех этих излияний кое-чего недостает. Я увидел точку зрения инженеров, программистов, гуру пользовательских интерфейсов. Однако проектировщики продуктов и маркетологи, то есть те люди, которые принимают ключевые решения по дизайну и проектированию аппаратного и программного обеспечения, демонстративно промолчали. Эй, ребята, у вас тут полно обозленных клиентов. Вам есть что на это ответить?

А в самом деле – вам есть что на это ответить?

Примечания

1

Считая от 2006 года, когда была издана книга. – *Примеч. ред*.

2

Aegis (англ. *эгида*) представляет собой интеграцию средств обнаружения, поражения и управления кораблем, целью которой является противостояние ракетным атакам противника. – *Примеч. пер*.

3

Честер Уильям Нимиц, главнокомандующий Тихоокеанским флотом США во время Второй мировой войны. – *Примеч. пер*.

4

В индустрии компьютеров под термином «разработчик программного обеспечения» понимается то же, что и под термином «программист»; аналогично следует воспринимать данные понятия и в этой книге. – *Примеч. авт*.

5

«Об интерфейсе», первое издание вышло в 1995 году, в 2003 году вышло второе издание, почти полностью переработанное. – *Примеч. ред*.

6

Меня постоянно пытаются убедить, что эта функция очень нужна женщинам, так как якобы может помочь избежать преступных нападений на неосвещенных парковках, но каждый раз я слышал это от технического специалиста мужского пола, которому эта кнопка никогда бы не пригодилась. К моему великому удивлению, не так давно я увидел в Wall Street Journal заметку о реальном случае применения этой кнопки с надписью Panic. Одна семья отдыхала в кемпинге в Йосемитском национальном парке, когда к их автомобилю, где были заперты съестные припасы, подошел дикий медведь и начал царапать его, пытаясь забраться внутрь. Тогда мать семейства нажала эту кнопку, и взвывшая сирена в итоге заставила медведя сбежать. В таком случае, вероятно, эту кнопку стоило бы назвать «Средство для отпугивания медведей».

7

Специалист по юзабилити (usability professional) – это не то же самое, что проектировщик взаимодействия (interaction designer). Подробно различия рассмотрены в главе 12 «В отчаянном поиске юзабилити».

8

Если быть точнее, мы утверждали, что хотим «сделать процесс объединения компьютеров на базе Intel/Windows таким же простым, как объединение компьютеров Macintosh». В те времена было до смешного просто сделать сеть из компьютеров Mac с помощью AppleTalk. И совсем не так просто сделать это с компьютерами Wintel, даже в настоящее время.

9

В некоторых почтовых клиентах есть возможность составления цепочек вручную и управления ими, только, как говорится, лекарство в иных случаях хуже самой болезни. Администрировать такие цепочки довольно сложно, и связанные сообщения по-прежнему считаются чем-то нетипичным.

10

Стоит признать, что я не лучше других разработчиков в этом смысле. В 1984 году я создал программу SuperProjects для *Computer Associates* – одну из первых программ по управлению проектами. Но в ней, как и во всех последующих программах, отсутствовала поддержка одновременного взаимодействия нескольких проектов.

11

Одна старая шутка из сферы компьютерной индустрии рассказывает, что департамент исследований *Microsoft* расположен в Купертино, намекая на то, что в этом же городе Кремниевой долины находится Центр перспективных разработок компании *Apple*.

12

И хотя я очень привержен проектированию, в такой тактике определенно есть смысл, и я ее поддерживаю. Если бы я нашел совершенно пустую рыночную нишу, я бы молниеносно и безжалостно попытался ввести продукт в игру. Тем не менее как только первая версия продукта увидит свет, я бы сосредоточился на тщательном проектировании второй версии. Если этого не сделаю я, то, несомненно, конкуренты могут обогнать меня в этом.

13

Как описывает Джеффри Мур в своей потрясающей книге *Crossing the Chasm* («Преодоление пропасти»), новые опции помогут привлечь только узкий круг ранних почитателей вашего продукта, но не более того.

14

Мишель Куин (Michelle Quinn), Vanishing Act («Фокус с исчезновением»), выпуск *San Jose Mercury News West Magazine* от 15 марта 1998 года.

15

Gerald Weinberg, *The Secrets of Consulting: A Guide to Giving & Getting Advice Successfully*(«Секреты консультирования: как с успехом давать и принимать советы»), Dorset House, 1985.

16

Здесь имеются в виду небольшие пакетики, которые вставлялись в нагрудный карман, где обычно носят ручку, и защищали рубашку от протекания чернил. Считаются одним из атрибутов культуры «ботаников» второй половины XX века. – *Примеч. пер*.

17

Po Bronson «The First $20 Million Is Always The Hardest», Avon Books, New York, 1997.

18

Ладно, стоит признать: я пилот частного самолета. Гари Килдалл – программист и настоящий фанат своего дела – в 1979 году позволил мне полетать с ним на его Piper Archer. После этого кратковременного полета я буквально «подсел» на это занятие. Программист внутри меня обожает всю эту сложность, доведенную до абсурда.

19

Также встречаются названия «крайние случаи», «аномальные ситуации», «пограничные ситуации».

20

Fred Moody, *I Sing the Body Electronic*, 1995, Viking, New York.

21

Paul Glen «Leading Geeks: How to Manage and Lead the People Who Deliver Technology», 2003, John Wiley & Sons, New York.

22

Для вас, издатели и любители латыни, будет небезынтересен тот факт, что у нас в Cooper Interaction Design денно и нощно кипят ярые баталии на тему верного написания этого термина: personas или же personæ. Сторонники первого варианта убеждают нас, что в таком случае слово будет читаться более понятно, лишние лигатуры исчезнут, и для восприятия клиентов такое написание окажется более удобным и менее устрашающим. Приверженцы второго варианта возражают, что способ чтения слова прояснится, стоит услышать его всего один раз, лигатуры – это манна небесная, а у наших клиентов достаточно ума, чтобы разобраться с загадками древнего языка. Для меня все это звучит так же, как споры программистов об алгоритмах, так что на страницах этой книги я буду придерживаться написания personas.

23

Фургон Конестога – крытый конный фургон, который использовался переселенцами из долины Конестога, штат Пенсильвания. – *Примеч. ред*.

24

Saul W. Gellerman «Motivation and Productivity», Amacom, New York, 1963.

25

Byron Reevs, Clifford Nass, *The Media Equation; How People Treat Computers, Television, and New Media Like Real and Places*, Cambrige University Press, 1996.

26

Steven Pinker, *How the Mind Works*, W. W. Norton & Company, 1997. Я очень люблю эту прекрасную, увлекательную книгу, написанную весьма компетентно. Довольно интересное чтение, раскрывающее глаза на многие факторы.

27

Игры являются примечательным исключением из этого правила. Многие игры просто не были бы такими увлекательными, если бы некоторые факты не были скрытыми, процессы неясными, а цели – размытыми.

28

Robert X. Cringely, *Accidental Empires, How the Boys of Silicon Valley Make Their Millions, Battle Foreign Competition, and Still Can't Get a Date*, Addison-Wesley, 1991.

29

На самом деле полный набор содержал больше персонажей, но самыми яркими оказались Бетси и Эрни.

30

Процесс создания сайтов – это так или иначе программирование, и повторное использование кода также обладает непреодолимой притягательностью и для Бетси.

31

Edward de Bono, *Lateral Thinking*, Creativity Step by Step («Латеральное мышление: Учебник творческого мышления»), 1970, Harper & Row, Publishers, New York.

32

Конечно, время и стремительное снижение цены на кремний значительным образом изменило конъюнктуру рынка сканеров. Данная информация актуальна на январь 1997 года.

33

«Частный детектив Магнум» – американский детективный телесериал 1980 года.

34

Я бы с превеликим удовольствием переделал в этой мощной и сложной программе весь интерфейс! Заметьте, как минимум шестеро из тех, кто читал эту рукопись, выделили эту сноску и добавили комментарии наподобие «Я бы тоже!» или «О да, сделайте это!».

35

Pee Wee Herman – комический псевдоним американского комедийного актера Пола Рубенса.

36

В своей книге под названием *About Face* («Алан Купер об интерфейсе») я привожу порядка 50 мощнейших аксиом проектирования. Это пример одной из них.

37

Это был конкурс в рамках отраслевой конференции COMDEX, он назывался Windows World Open и спонсировался такими компаниями, как Microsoft, Computerworld и Ziff-Davis Events. На тот момент конкурс проводился уже в седьмой раз.

38

Нумерация версий в данном случае абсолютно нелогична. До выпуска Windows 3.0 было как минимум четыре основных версии Windows, а Windows 3.1 имела кардинальные отличия и была значительно улучшена, так что ее точно следовало обозначить как Windows 4.0. Думаю, что такая нумерация была предложена маркетологами компании ввиду нежелания Microsoft упустить денежные средства, уже вырученные от «третьей версии».

39

David Maister «Managing the Professional Service Firm», 1997, Free Press, New York.

40

Stephen H. Wildstrom, «They’re Mad as Hell Out There», *Business Week*, October 19, 1998.