Чтобы пользователи наших продуктов работали более продуктивно, эффективно и увлеченно, нужно убедиться, что пользователь пребывает в нужном настрое. Обсудим способы сделать так, чтобы наши продукты оказывали поддержку человеческому интеллекту и способностям, и избегали разрушения того состояния продуктивной концентрации, в котором должны находиться пользователи.

Состояние потока

Люди, способные всецело сосредоточиться на некоторой деятельности, забывают о посторонних проблемах и отвлекающих факторах. Такое состояние называется **потоком**.

Поток можно описать как «состояние глубокого, почти медитативного погружения в работу». Поток нередко вызывает «мягкое чувство эйфории», так что человек, бывает, перестает замечать течение времени. Важно, что в состоянии потока человек может быть исключительно продуктивным, особенно если он занят созидательной работой, например конструированием, дизайном, разработкой или созданием литературных произведений.

Чтобы сделать людей более продуктивными и счастливыми, необходимо проектировать интерактивные продукты, вызывающие и поддерживающие состояние потока, и прикладывать все возможные усилия, чтобы избежать любого поведения, потенциально способного разрушить поток.

Если программа то и дело выбивает пользователя из потока, ему трудно возвращаться в это продуктивное состояние.

Если вы будете прежде всего думать об интерфейсе как таковом, на первый план выйдут побочные эффекты инструментов и технологий, а не цели пользователя. Когда вы в следующий раз будете ликовать по поводу замечательно спроектированного взаимодействия, вспомните, что, в конечном счете, **лучшим пользовательским интерфейсом в большинстве случаев является полное отсутствие интерфейса. Каким бы замечательным ни был ваш интерфейс, чем его меньше, тем лучше.**

Чтобы привести пользователя в состояние потока, вы должны сделать его взаимодействие с программным продуктом прозрачным. Когда романист пишет хорошо, писательское ремесло остается незамеченным, а читатель воспринимает только историю и ее персонажей с ясностью, которой техника письма не препятствует. Точно так же, когда продукт хорошо взаимодействует с человеком, механика взаимодействия исчезает, оставляя человека один на один с его целями без какой-либо посреднической функции программы. Плох тот писатель, читатели которого обращают внимание на текст, и плох тот проектировщик взаимодействия, чье неуклюжее присутствие ощущается в созданных им программах.

Не существует правил, благодаря которым предложение становится прозрачным. Все зависит от действий главного героя или от воздействия, которое желает оказать на читателя автор. Писатель понимает, что не следует использовать непонятные слова в особенно тихом и чувственном абзаце, так как они прозвучат как фальшивые ноты в струнном квартете.

То же верно и для программного обеспечения. Проектировщик взаимодействия должен учить себя слышать фальшивые ноты в **оркестровке** программного взаимодействия. **Жизненно важно, чтобы все элементы интерфейса работали скоординированно для достижения единой цели. Когда общение приложения с человеком организовано хорошо, оно становится практически незаметным.**

Оркестровку можно определить как «гармоничную организацию». Гармоничная организация не следует жестким правилам. Невозможно создать указания вроде «Пять кнопок в диалоговом окне – это хорошо» и «Семь кнопок в диалоговом окне – это уже слишком много». Однако легко догадаться, что не стоит создавать диалоговое окно с 35 кнопками. Серьезное затруднение, которое вызывают рассуждения такого рода, состоит в том, что проблема рассматривается *в искусственных условиях*. Подобный анализ не принимает во внимание задачу, которую требуется решить, равно как и то, чем занят в данный момент человек и чего он пытается добиться.

Проектирование гармоничного взаимодействия

Не существует универсальных правил, определяющих гармоничное взаимодействие, как не существует универсальных правил определения гармоничных пауз в музыке, однако мы находим, что перечисленные ниже стратегии направляют проектирование взаимодействия в нужное русло:

1. Следуйте ментальным моделям пользователей.
2. Меньше – лучше.
3. Позволяйте пользователю управлять, не принуждайте их к диалогу.
4. Держите инструменты под рукой.
5. Обеспечивайте немодальную обратную связь.
6. Проектируйте наиболее вероятное, будьте готовы к возможному.
7. Предоставляйте информацию о контексте.
8. Организуйте непосредственное манипулирование и графический ввод.
9. Отображайте состояния объектов и статус приложения.
10. Избегайте ненужных сообщений.
11. Не используйте диалоговые окна, чтобы сообщить, что все нормально.
12. Избегайте чистого листа.
13. Просите прощения, а не разрешения.
14. Отделяйте функции от их настройки.
15. Не задавайте вопросы – предоставляйте выбор.
16. Прячьте рычаги катапультирования.
17. Оптимизируйте скорость реакции; предупреждайте о задержках.

Следуйте ментальным моделям пользователей.

Разные пользователи имеют разные ментальные модели определенной деятельности или процесса, но эти модели редко представлены в терминах того, как на самом деле функционируют компьютеры. **Каждый пользователь естественным образом формирует в уме образ того, как программа выполняет свою работу. Мозг пытается найти какую-нибудь причинно-следственную связь между событиями и объяснить поведение компьютера.**

Например, в медицинской информационной системе медперсонал имеет ментальную модель базы данных о пациентах, основанную на медицинских картах, с которыми они сталкиваются в реальной жизни. Следовательно, имеет смысл реализовать поиск информации о пациенте по имени. Каждый врач наблюдает нескольких больных, так что имеет смысл дополнительно фильтровать пациентов в интерфейсе информационной системы так, чтобы врач мог выбирать пациента из списка, отсортированного по именам. С другой стороны, сотрудников больничной бухгалтерии в первую очередь интересуют неоплаченные счета пациентов. Изначально им все равно, как зовут больных, – важны сроки оплаты счетов (и, возможно, суммы). Значит, для бухгалтерии интерфейс информационной системы должен сортировать записи о пациентах по срокам задержки оплаты счетов и, возможно, по их суммам, а имена пациентов отходят на второй план.

Меньше – лучше.

Во многих случаях действует правило «чем больше – тем лучше». В области проектирования взаимодействия справедливо обратное утверждение, и **мы должны постоянно бороться за сокращение количества элементов интерфейса без сокращения возможностей продукта.** Чтобы добиться этого, мы должны сделать много, пользуясь минимумом средств, и здесь особенно важна тщательная оркестровка. Мы должны координировать и контролировать функциональность программного продукта, не позволяя интерфейсу превращаться в нагромождение диалоговых окон, усыпанных нелогичными и редко используемыми элементами управления.

Есть программы для бизнеса, в которых каждая функция или возможность «живет» в отдельном диалоге или окне, словно разработчики не задумывались, каким образом людям необходимо сочетать функции для решения определенных задач. Пользователям часто приходится выполнять команду меню, чтобы получить фрагмент информации, затем копировать эту информацию в буфер обмена, после чего в другом окне выполнять еще одну команду – и все это лишь для того, чтобы вставить скопированную информацию в поле ввода. Это не просто неэлегантное и грубое решение – это подход, провоцирующий ошибки и не способный поддерживать продуктивное разделение труда между человеком и машиной. Как правило, подобные продукты создаются неспециально – это получается в ходе многолетней разработки с принятием решений на ходу или вследствие разработки продукта несколькими изолированными командами, обитающими в отдельных резервациях внутри организации.

Важно помнить, что **стремление к простоте может зайти слишком далеко – сокращение интерфейса есть поиск баланса, требующий хорошего понимания ментальных моделей пользователей.**

Позволяйте пользователям управлять, не принуждайте их к диалогу.

Многие разработчики, похоже, воображают, что идеальный интерфейс должен иметь форму диалога с пользователем. Однако пользователи в большинстве своем так не думают. Они предпочитают взаимодействовать с программой примерно так же, как с автомобилем: они открывают дверцу и садятся в машину, когда хотят куда-нибудь съездить; они давят на педаль газа, когда хотят, чтобы машина ехала, и на тормоз, чтобы она остановилась; они поворачивают руль, когда хотят повернуть.

Такое идеальное взаимодействие не является диалогом. Скорее, это использование средства передвижения.

Одной из причин, по которым программный продукт нередко раздражает пользователей, является то, что **он часто вовлекает нас в диалог: сообщает нам о наших промахах и требует от нас ответа. С точки зрения пользователей, все должно быть наоборот: пользователь требует, а компьютер отвечает.**

Держите инструменты под рукой.

Приложение обычно предлагает пользователю набор разнообразных инструментов. Предоставление инструментальных средств потенциально усложняет программу, и требуется затратить еще много сил, чтобы манипулирование инструментом стало действительно простым и не выводило пользователя из состояния потока. В первую очередь мы должны обеспечить полноту и доступность информации о текущем инструменте, а также быстроту и легкость переключения с одного инструмента на другой.

Инструменты должны быть под рукой – предпочтительно на палитре или на панели инструментов, а также в виде клавиатурных сокращений – для специалистов. Тогда пользователь будет их видеть и сможет одним щелчком или нажатием клавиши выбрать любой из них. **Если пользователь вынужден переключать внимание с основного занятия на поиски нужного инструмента, он утратит сосредоточенность.** Это все равно что встать из-за стола и пойти в другой конец дома за карандашом.

Обеспечивайте немодальную обратную связь.

Когда пользователь интерактивного продукта манипулирует инструментами и данными, как правило, желательно, чтобы программа сообщала ему об их состоянии и о результатах произведенных манипуляций. Эта информация должна быть легко читаемой и понятной, не заслоняющей элементы интерфейса и не препятствующей нормальной работе пользователя.

Приложение имеет ряд способов представлять информацию или осуществлять обратную связь с пользователем. К сожалению, самым распространенным подходом является вывод диалогового окна. Эта техника модальна: программа переходит в специальный режим, требующий реакции пользователя, без которой она не может вернуться в нормальное состояние, а пользователь не может продолжать работу. **Более удачным подходом является обеспечение немодальной обратной связи.**

Обратная связь *немодальна*, если информация для пользователя включена в основной интерфейс и не прерывает нормальную работу системы и ее взаимодействие с пользователем. Например, редактор Word сообщает вам номер текущей страницы и раздела, количество страниц в документе, позицию курсора и текущее время. Вся эта информация немодальна: вы просто смотрите на строку состояния внизу окна, и для получения информации вам не требуется выполнять сложные действия.

В кабине истребителя есть специальное устройство, которое выводит показания важнейших приборов прямо на лобовое стекло. Пилоту да же не приходится пользоваться периферийным

зрением: он видит всю необходимую информацию, не отрывая взгляда от самолета противника.

Проектируйте наиболее вероятное, будьте готовы к возможному.

Существует много случаев, когда взаимодействие с пользователем (обычно в форме диалоговых окон) проникает в интерфейс без особой необходимости. Часто это происходит, когда программа стоит перед выбором. Большинство программистов решают проблему выбора с позиций логики, и это отражается на создаваемом продукте. **С точки зрения логики, если утверждение справедливо в 999 999 случаях из миллиона и ложно в одном случае, то оно ложно. Так работает булева логика. Однако для большинства людей оно будет безусловно истинным.** Утверждение *может* быть ложным, но *вероятность* этого мала на столько, что ею можно пренебречь.

Программисты обычно считают, что возможность и вероятность – одно и то же. Например, пользователь может завершить работу с программой и сохранить свою работу, а может выбросить документ, над которым работал в течение шести часов. Любой поворот событий возможен. Вероятность того, что пользователь выбросит результаты своего труда, – не больше, чем одна тысячная. Тем не менее типичная программа содержит диалоговое окно с вопросом, не хочет ли пользователь сохранить изменения.

Представляйте информацию с учетом контекста.

Способ представления информации приложением – это еще один путь для создания помех в сознании пользователя. Особые злоупотребления наблюдаются при представлении количественной, то есть цифровой информации. Если приложение должно сообщить пользователю об объеме свободного пространства на диске, оно может вывести *точное* количество свободных байтов.

Эти числа трудно воспринимать и трудно интерпретировать. Когда диск содержит больше десятка миллиардов байтов, для нас уже неважно, сколько сотен байтов свободно. Тем не менее программа подсчитывает килобайты. Несмотря на такую точность, передача информации затруднена. В действительности пользователь хочет знать, заполнен ли диск до отказа – или можно добавить программу размером 20 Мбайт и еще останется достаточно места. Голые цифры, какими бы точными они ни являлись, мало помогают в этом вопросе.

Организуйте непосредственное манипулирование и графический ввод.

Программные продукты редко представляют численную информацию в наглядном виде. Еще реже они позволяют вводить данные в графической форме. Многие программы дают возможность вводить числа, а затем – по команде – преобразуют эти числа в графическое представление. Редкие продукты позволяют пользователю создать график и по команде преобразовать его в последовательность чисел. Исключение – современные текстовые процессоры: почти каждый процессор позволяет указывать позиции табуляции и отступы путем перетаскивания маркера на линейке. Пользователь как бы говорит: «Я хочу, чтобы абзац начинался здесь», – и позволяет программе вычислить, что отступ равен 1,347 дюйма от левого поля. К счастью, пользователь не обязан вводить число 1,347 вручную.

Этот принцип действует в самых разных ситуациях. Когда нужно упорядочить элементы какого-либо списка, пользователь, возможно, за хочет расположить их в алфавитном порядке. Однако он может захотеть расположить их по своему вкусу, а не в соответствии с неким алгоритмом. У пользователя должна быть возможность перетаскивать элементы списка непосредственно, без вмешательства каких бы то ни было алгоритмов.

Отображайте состояния объектов и статус приложения.

Когда человек спит, он выглядит спящим. Когда он проснулся, он выглядит бодрствующим. Если человек занят делом, он и выглядит соответственно: его взгляд сосредоточен на объекте работы, а поза является закрытой и демонстрирует занятость. Люди не просто ожидают друг от друга подобной обратной связи; поддержание общественного порядка приводит к определенной зависимости от этой обратной связи.

Наши приложения и устройства должны вести себя аналогично. Когда программа «спит», она должна выглядеть спящей. Приложение в активном состоянии и должно выглядеть активным, а приложение, выполняющее определенные действия, должно выглядеть соответственно. Когда компьютер занят важной работой, например выполняет сложные вычисления или подключается к базе данных, должно быть очевидно, что приложение не будет так шустро реагировать на наши запросы, как обычно. Когда компьютер отправляет факс, мы должны видеть анимацию сканируемого и отправляемого факса (или хотя бы немодальный индикатор хода операции).

Отображать состояние программы и объектов лучше всего с помощью обогащенной немодальной обратной связи.

Избегайте ненужных сообщений.

Программистам необходимо точно знать, что происходит в программе. Для них это связано с возможностью детального контроля внутренних процессов. Зато пользователям нет никакого дела до мелких подробностей того, что творится внутри программы. Например, люди, не разбирающиеся в технике, могут быть встревожены сообщением, что база данных была модифицирована. Гораздо лучше, когда программа просто выполняет свою работу, сообщает об успешном завершении операций и не нагружает пользователя подробностями о том, *как* она добилась результата.

Многие приложения усердно держат пользователя в курсе всех подробностей своей работы, даже если он не имеет ни малейшего представления о том, что делать с этой информацией. Программы выводят диалоговые окна, сообщающие нам, что соединение установлено, записи отправлены, пользователи зарегистрированы в системе, транзакции записаны, данные переданы, – и уйму других бесполезных фактов. Для программистов эти сообщения свидетельствуют о том, что все благополучно. По всей видимости, эти сообщения использовались при отладке программы. Однако для обычного человека они равносильны тревожным огням на горизонте.

Как было сказано ранее, **приложение должно недвусмысленно сообщить пользователю, что выполняет некую работу, но подробная обратная связь должна быть организована более тонко.** В частности, вывод сведений, перечисленных выше, в форме диалоговых окон

прервет взаимодействие пользователя с программой, не предложив взамен ни какой компенсации.

Важно не прерывать работу ради выдачи сообщения о том, что все протекает нормально. Когда происходит ожидаемое событие, не нужно сообщать о нем с помощью диалогового окна. Поберегите диалоговые окна для событий, выходящих за рамки нормального положения дел.

Не используйте диалоговые окна, чтобы сообщить, что все нормально.

По сходным причинам не следует прерывать работу, чтобы побеспокоить пользователя сообщением о несерьезных проблемах. Если программа не смогла создать соединение с сервером, не выводите ради этого диалоговое окно. Поместите в окне программы индикатор состояния, чтобы заинтересованный пользователь знал, что происходит, а пользователь, занятый другими делами, не отвлекался.

Секрет оркестровки взаимодействия с пользователем заключается в подходе, ориентированном на цели. Вы должны спросить себя, способно ли конкретное взаимодействие быстро приблизить пользователя непосредственно к его цели. Современные приложения нередко отказываются делать хоть чтото самостоятельно, без команды пользователя. Однако пользователь предпочел бы, чтобы приложение предприняло разумный первый шаг, который потом можно было бы скорректировать. Так программа приблизила бы к его цели.

Избегайте чистого листа.

Гораздо проще не задумываться о том, чего хочет пользователь, а за дать ему кучу вопросов и принять решение на основе его ответов. Однако *нормальные* люди чувствуют себя неуютно, когда им приходится объяснять интерактивному продукту, чего они хотят. Они предпочли бы, чтобы программа сделала то, что, *по ее мнению*, следует сделать, а за тем скорректировали бы результат. В большинстве случаев программа в состоянии сделать правильное предположение на основании предыдущего опыта. Например, когда вы создаете новый документ в Microsoft Word, то получаете пустой документ с конкретными отступами и прочими атрибутами, а не оказываетесь лицом к лицу с диалоговым окном, требующим указать все эти детали.

Просите прощения, а не разрешения.

Из того, что мы часто применяем к интерактивным продуктам слово *думает*, вовсе не вытекает, что программа обязана быть интеллектуальной (как это слово понимают люди) и пытаться выработать правильную линию поведения посредством рассуждений. Она просто **должна опираться на статистику и совершать действия, правильность которых весьма вероятна, а затем предоставлять пользователю развитые инструменты для корректировки первой попытки.** Это гораздо лучше, чем выдавать пользователю чистый лист, вынуждая его заполнять этот лист самостоятельно. В результате программа не просит разрешения действовать, но просит прощения за уже содеянное.

Для большинства людей совершенно пустой лист является неудобной отправной точкой. Им гораздо легче начать, если на нем уже что-то написано. Пользователь с готовностью

подкорректирует предложенный программой черновик, лишь бы избавить себя от умственных усилий, необходимых при разработке проекта с нуля.

Отделяйте функции от их настройки.

Еще одна проблема довольно часто возникает, когда пользователи вызывают функции с большим количеством параметров. Причина проблемы кроется в неспособности разработчиков различать функцию и *настройку* этой функции. **Если вы просите приложение выполнить саму функцию, приложение должно просто выполнить ее, а не допрашивать вас о подробностях настройки. Если вы захотите уточнить свои требования, то вызовете диалоговое окно для настройки.**

Например, в ответ на просьбу пользователя распечатать документ многие программы открывают сложное диалоговое окно, требующее указать количество копий, ориентацию бумаги, лоток принтера, размеры полей, цветной или монохромной должна быть печать, какой нужен масштаб, использовать ли встроенный шрифт или шрифт PostScript, печатать ли весь документ, текущую страницу либо только выделенный фрагмент, следует ли печатать в файл, и если да, то какое имя будет у этого файла. Все эти возможности полезны, но ведь мы хотели всего лишь распечатать документ и просили только об этом.

Более осмысленный интерфейс включал бы в себя команду печати и отдельную команду для настройки печати. Первая должна обходиться без диалоговых окон и всего лишь выполнять печать, придерживаясь либо последних настроек, либо настроек по умолчанию. Функция настройки печати предложит пользователю все те варианты выбора, связанные с бумагой, шрифтами и количеством копий, что были пере числены выше. Будет вполне логично, если окно настройки позволит тут же распечатать документ.

Существует огромная разница между настройкой и вызовом функции. **Настройка, в принципе, может и не включать выполнение функции, но вызов функции уж точно не должен включать в себя настройку.** Вообще говоря, любой пользователь вызывает команду в десять раз чаще, чем настраивает ее. Пусть лучше он запросит диалоговое окно настройки один раз из десяти, чем будет отказываться от интерфейса настройки *девять* раз из десяти.

Не задавайте вопросы – предоставляйте выбор.

Задавать вопросы – совсем не то же самое, что предоставлять выбор. Разница между этими действиями такая же, как между собеседованием с кандидатом на должность и выбором товаров на полках супермаркета. Человек, задающий вопросы, оказывается в положении, более высоком по отношению к тому, кого он опрашивает. Начальство спрашивает, подчиненные отвечают. Программа, задающая вопросы, заставляет пользователя ощущать себя подчиненным и вызывает у него раздражение.

Диалоговые окна (особенно диалоги подтверждения) задают вопросы. Панели инструментов предоставляют выбор. Диалоговые окна подтверждения прерывают работу, требуют ответа и не уходят, пока не получат то, чего хотят. Панели инструментов всегда присутствуют на экране, тихо и вежливо предлагая то, что у них есть, подобно хорошо организованному супермаркету, предоставляя клиенту роскошь любо го выбора одним движением пальца.

В противоположность тому, что думают многие разработчики программ, вопросы и возможность выбора далеко не всегда вызывают у пользователя ощущение собственного могущества. Гораздо чаще пользователь чувствует себя затравленным и встревоженным.

Пользователи не любят, когда им задают вопросы. У них создается впечатление, что программа:

1. невежественна;
2. забывчива;
3. слаба;
4. безынициативна;
5. не способна позаботиться о себе;
6. капризна;
7. излишне требовательна.

Обычно мы осуждаем эти качества в людях – так почему мы должны терпеть их в программном продукте? В отличие от друга за ужином в ресторане, программа задает нам вопросы не потому, что ей просто интересно наше мнение, и не из желания поддержать разговор. Она ведет себя как человек невежественный и пытающийся преувеличить свою значительность. Программа не интересуется нашим мнением – она требует информацию, причем информацию не первоочередной важности.
Хуже однократных вопросов – вопросы, задаваемые многократно и без необходимости. Многие банкоматы постоянно спрашивают у пользователей о предпочтительном языке: «Испанский, английский или китайский?». Но вряд ли этот ответ изменится, когда пользователь освоится с банкоматом. Программа, задающая меньше вопросов, покажется пользователю более умной, вежливой и внимательной.
Люди обращаются с компьютерами и другими интерактивными устройствами, *как с людьми*, и реагируют на них, *как на людей*. Следовательно, мы должны позаботиться о качествах «личности», которую видят пользователи в нашей программе.

Прячьте рычаги катапультирования.

В кабине каждого истребителя есть ярко окрашенный рычажок. Если за него потянуть, небольшой реактивный двигатель под сиденьем пилота сработает и вытолкнет пилота вместе с сиденьем из самолета, после чего раскроется парашют – и пилот благополучно спустится на землю. Рычажок катапульты можно использовать только один раз, а последствия его использования значительны и необратимы.

Катапульта в истребителе необходима, так же как средства настройки необходимы в сложных настольных приложениях. Рычаг катапульты нужен, но используется он нечасто.

Программы должны иметь «катапульты», чтобы пользователи могли время от времени перемещать внутри интерфейса *стабильные объекты* или существенно (иногда необратимо) менять функциональность либо поведение приложения. При проектировании интерфейса

следует убедиться, что пользователь не сможет нечаянно катапультироваться, когда ему требуется лишь слегка подкрутить настройки программы.

Рычаги «катапульты» бывают двух типов. Одни сильно изменяют внешний вид программы (расположение инструментов и рабочих областей), а другие выполняют какие-либо необратимые действия. Обе эти функции должны быть спрятаны от неопытных пользователей. Второй тип наиболее опасен. В первом случае пользователь может быть удивлен или напуган результатом, но он, по крайней мере, сможет вернуться к прежнему состоянию после некоторых усилий. Во втором случае и пользователю, и его коллегам придется смириться с последствиями.

Оптимизируйте скорость реакции; предупреждайте о задержках.

Обрабатывая большой объем информации или общаясь с удаленными серверами, принтерами, сетями, приложение может замедлить свою работу или перестать реагировать на действия пользователя. Нет ничего более опасного для состояния потока пользователя, чем необходимость смотреть на экран, ожидая ответа компьютера. **Крайне важно проектировать интерфейсы таким образом, чтобы они достаточно быстро реагировали на команды пользователя** – ни один «гламурный» интерфейс на свете не способен впечатлить пользователя, когда он работает со скоростью улитки из-за интенсивной отрисовки экрана.

Здесь важно сотрудничать с разработчиками. Различного рода взаимодействие может, в зависимости от платформы и технологической среды, иметь достаточно высокую стоимость с точки зрения скорости реакции приложения. Вы должны не только выступать за интерфейс, обеспечивающий пользователя насыщенным взаимодействием с минимальными задержками, но и создавать решения, учитывающие выбор технической платформы в начале проекта, который нельзя изменить. Когда задержки неизбежны, важно четко сообщить об этом пользователю и дать ему возможность отменить операцию, вызывающую задержку, а в идеале – еще и возможность делать другую работу во время ожидания.

Если приложение выполняет потенциально «долгоиграющие» задачи, не забывайте время от времени проверять, что делает пользователь. Возможно, он барабанит по клавиатуре или бешено щелкает мышью, завывая при этом: «Да нет же, не хотел я перестраивать *всю* базу данных. Это же займет 4,3 миллиона лет!»

В ряде исследований, проведенных еще в конце 60х годов, ученые выяснили, что восприятие пользователем времени реакции можно грубо разделить на несколько интервалов:

1. **До 0,1 секунды пользователи воспринимают отклик системы как моментальный.** Они чувствуют, что напрямую манипулируют пользовательским интерфейсом и данными.

2. **До 1 секунды пользователи чувствуют, что система реагирует.** Вероятно, они замечают задержку, однако эта задержка недостаточно велика, чтобы прервать мыслительные процессы.

3. **До 10 секунд пользователи замечают, что система работает медленно, и отвлекаются, однако способны сохранять некоторое внимание к приложению.**

Здесь важно наличие индикатора хода работы.

4. **После 10 секунд внимание пользователя полностью рассеивается.** Он уходит за чашкой кофе или переключается на другое приложение. В идеале такие длительные процессы должны проводиться в фоновом режиме или без участия пользователя, позволяя ему заняться другой работой. В любом случае следует четко обозначать состояние и ход процесса, в том числе оставшееся время. И просто обязателен механизм отмены.

В общем, создание успешного продукта требует не только предоставления полезных функций. Следует задуматься также об оркестровке различных функциональных элементов, позволяющей пользователям достигать состояния потока при решении своих задач. **Лучшие пользовательские интерфейсы не поражают пользователей своей красотой – скорее они почти незаметны, поскольку с ними крайне легко работать.**

Оптимизация налогообложения

Программным средствам часто свойственны взаимодействия, вынуждающие пользователя выполнять дополнительную работу. Программисты, как правило, настолько сосредоточены на технологиях, лежащих в основе продукта, что не особенно вникают в действия, которые требуются от человека. В результате получается приложения, взимающее с пользователя налог - в виде умственного, а иногда и физического напряжения во время работы.

Любая крупная задача, например поездка на работу, распадается на ряд подзадач. Некоторые из них направлены непосредственно на достижение цели, например управление автомобилем на дороге. Налоговые задачи, с другой стороны, не приближают нас к цели, но все равно необходимы для ее достижения. Таким образом, **налоги — это работа, удовлетворяющая потребности либо наших инструментов, либо внешних агентов, с которыми мы сталкиваемся, пытаясь достичь цели.**

Когда речь заходит о программных продуктах, разделение подзадач на непосредственно ведущие к цели и налоговое бремя достаточно очевидно. Проблема с налоговыми задачами заключается в том, что усилия, направленные на их решение, не способствуют достижению цели. Когда нам удается сократить налоги, мы делаем работу наших пользователей более эффективной и продуктивной, а так же повышает пригодность нашей программы к использованию, создавая более качественный опыт взаимодействия с ней.

Сокращайте налоговое бремя при любой возможности.