

Глава 2

Проблемно-центрированная разработка интерфейса

Одним из наиболее эффективных подходов к разработке интерфейса с пользователем, предлагаемых в литературе по человеко-машинному взаимодействию, является подход, сфокусированный на задачах, которые нужно решать пользователю – проблемно-центрированный подход. Он противопоставляется так называемому "методу водопада", который ещё широко применяется и будет кратко охарактеризован в конце главы. При проблемно-центрированном подходе процесс разработки структурируется исходя из специфических задач, которые пользователь должен будет решать с помощью разрабатываемой системы. Эти задачи выбираются на ранней стадии разработки, затем они используются для выявления требований к дизайну, чтобы облегчить выработку решений и их оценку по мере развития проекта. Вот основные этапы проблемно-центрированного дизайна:

- анализ задач и пользователей;
- выбор репрезентативных задач;
- заимствование;
- черновое описание дизайна;
- обдумывание дизайна;
- создание макета или прототипа;
- тестирование дизайна с пользователями;
- итерирование;
- реализация;
- отслеживание эксплуатации;
- изменение дизайна.

Теперь рассмотрим предлагаемые этапы более подробно.

2.1. Анализ задач и пользователей

На этом этапе предстоит дать ответ на вопрос, кто и зачем собирается использовать разрабатываемую систему. Необходимость в анализе задач очевидна: построение большой системы, которая не делает того, что нужно, вряд ли можно назвать успехом дизайна. Но поверх простого требования "делать то, что нужно" хорошая система должна гладко входить в существующий мир пользователя и его работу. В частности, система должна запрашивать от пользователя информацию в порядке, который кажется ему естественным, она должна давать возможность простой коррекции ошибок при вводе данных, выбранные для организации интерфейса аппаратные средства должны вписываться в рабочую среду пользователя и быть эргономичными для его действий. Рассмотрение этих и множества других деталей интерфейса часто отсутствует при традиционной разработке, но они могут быть раскрыты, если дизайнер уделит время деталям задач, которые в действительности должен решить пользователь.

Важно также понимание пользователей как таковых. Осведомлённость об уровне знаний пользователя позволяет дизайнеру ответить на вопросы о выборе названий пунктов меню,

о том, какие материалы включить в обучающий комплект и контекстную помощь, и даже о том, какие возможности должна обеспечивать система. Если, например, система разрабатывается для пользователя персонального компьютера, привыкшего к приёму "копировать–вставить", то целесообразно, чтобы в новой системе этот приём также работал, даже если он не слишком важен с точки зрения основной задачи. Такие трудно оцениваемые различия среди пользователей, как уровень их квалификации, интерес к изучению новых систем, заинтересованность в успехе разработанной системы, могут обуславливать многие решения дизайнера, например, какой заложить уровень обратной связи с пользователем, где предпочесть команды, подаваемые с помощью клавиатуры, а где – выбираемые из меню и т.д.

Эффективный анализ задач и пользователей требует тесного персонального контакта между командой разработчиков и теми людьми, которые в дальнейшем будут пользоваться системой. Это не всегда просто осуществить. Часто руководители коллективов (как разработчиков, так и пользователей) становятся препятствием. Например, от разработчиков могут требовать представление существенных деталей проекта перед тем, как он будет показан заказчику. Однако несомненно, что ранний и непрерывный контакт между разработчиками и пользователями существенен для успеха разработки.

2.2. Выбор репрезентативных задач

После выработки хорошего понимания пользователей и их задач более традиционный процесс разработки может абстрагироваться от этих фактов и привести к общей спецификации системы и её пользовательского интерфейса. Проблемно-центрированный дизайн предлагает более жёсткий подход. Разработчик должен выделить несколько репрезентативных задач, которые будут решаться при использовании системы. Это должны быть задачи, которые пользователи описали разработчикам. Первоначально они могут быть описаны буквально в нескольких словах, но, т.к. речь идёт о реальных задачах, в любой момент в дальнейшем эти описания могут быть расширены до любой степени детальности, чтобы ответить на любые вопросы, касающиеся дизайна интерфейса или анализа предложенных решений. Вот несколько примеров:

- для программы обработки текстов: "извлечь запись мемо и поместить её в список рассылки";
- для электронной таблицы: "рассчитать бюджет заработной платы на следующий год";
- для коммуникационной программы: "войти на сервер организации через модем";
- для промышленной системы управления: "передать управление следующей смене".

Отметим ещё раз, что это должны быть реальные задачи, с которыми сталкиваются пользователи. Разработчики должны собрать все необходимые материалы для выполнения этих задач: копию файла с мемо-записью, информацию о заработных платах в текущем году и факторы, вызывающие их изменения, и т.д.

Отобранные задачи должны достаточно полно покрывать всю функциональность системы, и дизайнеру полезно сделать проверочный список всех функций и путём сопоставления его с перечнем задач удостоверится, что желаемое покрытие достигнуто. Это также должна быть смесь простых и более сложных задач. Простые задачи, например, "проверить правильность написания слова", полезны на ранних стадиях проектирования, но многие проблемы построения интерфейса будут выявлены только через сложные задачи, отражающие взаимодействия в реальном мире. Способность

разработчика произвести эффективный набор задач – это настоящий тест на его понимание пользователей и их работы.

2.3. Заимствование

На данном этапе очень полезно найти существующие интерфейсы, с помощью которых пользователи могут выполнить требуемую работу, и затем строить идеи новой системы на их базе насколько это законно и возможно практически. Этот сорт копирования может быть эффективным как для высокоуровневых парадигм взаимодействия, так и для низкоуровневых решений на основе специальных органов управления и дисплеев.

На высоких уровнях думайте о репрезентативных задачах и пользователях, которые их решают. Какие программы эти пользователи или люди, находящиеся в подобных ситуациях, используют сейчас? Если они используют электронные таблицы, то, может быть, ваш дизайн должен выглядеть как электронная таблица. Если они используют объектно-ориентированные графические пакеты, может быть, ваше приложение должно быть похоже на них. Вы можете быть способны создать новую парадигму взаимодействия, которая лучше подойдёт к разрабатываемому приложению, но риск неудачного дизайна довольно велик. Существующую парадигму реализовать быстрее и проще, т.к. многие проектные решения уже оказываются выполненными. Но что более важно, её легче будет изучить и удобнее применять для пользователей, т.к. они уже заранее будут знать, как работает большая часть интерфейса.

Копирование известных решений также полезно для низкоуровневых деталей интерфейса, таких как размещение кнопок и названия полей меню. Приведём пример. Пусть вы разрабатываете специализированный пакет управления формами, и его спецификация требует, чтобы в какой-то момент могла быть выполнена проверка правописания. Тогда вы должны посмотреть на элементы управления в подсистемах проверки правописания в текстовых процессорах, которые в данный момент используют будущие пользователи вашей системы. Будет чрезвычайно хорошо, если в вашей системе данный интерфейс будет построен таким же образом.

На пути заимствования многие дизайнеры допускают неверные решения по причине недостаточно глубокого видения деталей существующих реализаций поверх задач собственной разработки. Углубимся немного в пример с проверкой правописания. Допустим, ваш анализ показал, что программа проверки правописания обычно находит неправильно написанные слова, в частности, имена и фамилии, и может автоматически скорректировать их написание, используя базу данных пользователя. Вы решаете, что наилучшим вариантом взаимодействия будет высветить на дисплее предлагаемый вариант корректировки и позволить пользователю принять его путём нажатия на клавишу <Enter>. Но текстовый процессор, которые будущие пользователи вашей системы используют наиболее часто, следует другому соглашению: нажатие клавиши <Enter> сохраняет "неправильное" написание слова. Стоит ли следовать этому существующему правилу или реализовать своё, более эффективное? В известной степени решение зависит от того, будут ли пользователи чаще пользоваться вашей системой или текстовым процессором. Но чаще всего наилучшим вариантом будет придерживаться правила старой системы. Это позволяет пользователям применять наработанные навыки, хотя и требует при работе на одно или два нажатия клавиши больше, чем новый "оптимизированный" вариант.

2.4. Черновое описание дизайна

Черновое (грубое) описание разрабатываемой вами системы должно быть положено на бумагу (обязательно). Это позволяет задуматься о многих вещах. Но это описание не следует оформлять в виде компьютерной программы (пока), даже если вы умеете пользоваться какими-либо системами автоматизации разработки. Такие системы вынуждают вас прикрепляться к конкретным решениям, которые ещё слишком рано делать.

На этой стадии команда разработчиков может проводить множество дискуссий по поводу того, какие возможности должна включать в себя система и как они должны представляться пользователям. Дискуссия должна следовать проблемно-центрированному подходу. Если кто-то из команды предлагает введение новой возможности, то другой член команды обязан спросить его, решению какой из репрезентативных задач эта новая возможность будет способствовать. Возможности, которые не способствуют решению любой из задач, как правило, должны отбрасываться. Либо же список репрезентативных задач должен быть дополнен реальной задачей, которая требует этой возможности программы.

Репрезентативные задачи должны использоваться как контрольный список, позволяющий удостовериться, что описание системы полно. Если вы не можете представить решение каждой задачи внутри текущего определения системы, это определение должно быть доработано и улучшено.

2.5. Обдумывание дизайна

Никакая авиационная компания не будет разрабатывать и строить реактивный самолёт без предварительного инженерного анализа, который предсказывает основные технические характеристики. Стоимость строительства и риск неудачи слишком велики. Точно так же стоимость построения законченного пользовательского интерфейса и его тестирование с достаточным количеством пользователей для выявления всех проблем неприемлемо высока. Хотя разработка интерфейсов ещё не достигла такого уровня сложности, как авиационная инженерия, существует несколько структурных подходов, которые можно использовать, чтобы исследовать сильные и слабые стороны интерфейса до его программного воплощения.

Один из методов состоит в подсчёте количества нажатий клавиш, движений мыши и мыслительных операций (решений), необходимых для выполнения задач, предписанных разрабатываемой системе. Это позволяет оценить трудоёмкость выполнения задач по времени и выявить задачи, требующие слишком много шагов. Процедуры для реализации этого подхода, названные анализом GOMS, разработаны достаточно детально. Основные идеи GOMS будут рассмотрены позднее.

Другой метод основан на приёме, названном познавательный сквозной контроль (cognitive walkthrough, CWT), и позволяет находить места в дизайне, где пользователь может делать ошибки. Как и GOMS, CWT анализирует взаимодействия пользователей с интерфейсом при решении отдельных задач. Данный метод также будет представлен позже.

2.6. Создание макета или прототипа

После этапа обдумывания дизайна по его описанию на бумаге, приходит пора построить что-то более конкретное, могущее быть показанным пользователям и представляющее дальнейшую детализацию предстоящей работы. При разработке простых систем этот более конкретный продукт может быть просто серией рисунков на бумаге, показывающих вид интерфейса по мере того, как пользователь проходит шаги решения репрезентативных задач. Удивительно много информации можно получить, просто показав макет на бумаге нескольким пользователям. Макет даже может выявить скрытое непонимание между членами команды разработчиков.

Для более тщательного анализа и для более сложных систем могут использоваться специальные инструменты для создания прототипов. Можно даже построить прототип с помощью так называемых UIMS- систем (от англ. User Interface Management System – система управления интерфейсом пользователя), которые дадут фундамент для окончательного продукта. Этот подход может быть особенно продуктивным не только потому, что он уменьшает объём работы, требующейся для построения финального продукта, но и потому, что интерфейсные приёмы, протестированные на базе самостоятельного макета, могут оказаться трудными в конечной реализации. Рассмотрение профессиональных UIMS-систем, такие как Hypercard или ProcSee, выходит за рамки нашего курса. Для учебных целей в качестве UIMS-системы может рассматриваться визуальная среда программирования, создаваемая такими инструментами, как Borland Delphi, C Builder или MS Visual Studio. Хотя они и не считаются полноценными UIMS-системами, но в какой-то степени решают похожие задачи – отделение графической части интерфейса от логики функционирования системы.

На данном этапе не нужно реализовывать систему целиком. Усилия должны быть сосредоточены на частях её интерфейса, нужных для решения репрезентативных задач. Скрываемая за интерфейсом функциональность системы, которая может находиться ещё в стадии разработки, может эмулироваться методами "Волшебника из страны Оз". То есть сам разработчик или его коллега могут выполнять действия, которые система пока не может делать. Разумеется, не следует вводить в заблуждение пользователей, особенно их руководителей, чтобы они думали, что система уже завершена.

2.7. Тестирование дизайна с пользователями

Опыт показывает, что независимо от того, как тщательно был сделан анализ дизайна на предыдущих этапах, существуют проблемы, которые выявляются только при тестировании дизайна с пользователями. Тестирование должно проводиться с людьми, чей уровень образования и специальной подготовки примерно соответствует тому, что будет у реальных пользователей системы. Следует попросить пользователей выполнить одну или несколько репрезентативных задач, решение которых поддерживает система. Здесь оказывается эффективным приём "думанье вслух". Вы просите пользователя не только делать необходимые действия для решения поставленной задачи, но и говорить, что он делает и о чём размышляет. Можно полностью записывать его комментарии (например, на диктофон) или только делать заметки. Эти комментарии дают неоценимые данные для улучшения дизайна системы.

Информация, собранная при тестировании системы с пользователями, даёт ответ на многие вопросы: сколько времени потребовало выполнение тех или иных действий и

задач в целом, какие допускались ошибки, что вызвало затруднение или удивление у пользователя, даже если это и не привело к ошибке. Записанные комментарии пользователя позволяют понять, почему были допущены ошибки. Без комментариев вы только фиксируете сам факт ошибки, но вынуждены потом догадываться (додумывать за пользователя), почему это произошло. При анализе действий пользователя и его комментариев может выясниться, что он мыслит не так, как дизайнер системы. Это позволит внести корректировки, позволяющие приблизить интерфейс системы к её предполагаемому пользователю.

2.8. Итерирование

Тестирование с пользователями всегда показывает какие-то проблемы с дизайном. Помните, что цель тестирования состоит не в том, чтобы доказать правильность интерфейса, а в том, чтобы улучшить его. Необходимо проанализировать результаты тестирования, соизмеряя стоимость корректировок с серьёзностью возникших проблем, затем доработать интерфейс и протестировать его снова. Серьёзные проблемы могут даже потребовать пересмотра понимания задач и пользователей, т.е. откатить вас на первый этап проблемно-центрированного подхода.

Одна вещь, о которой необходимо помнить на каждой итерации – различные возможности и особенности интерфейса не самостоятельны. Например, переделывание меню для устранения проблемы, возникшей при выполнении одной задачи, может создать проблемы для других задач. Некоторые из таких взаимовлияний могут быть найдены при анализе без пользователей с помощью CWT или аналогичных приёмов. Другие же не выявятся без тестирования с пользователями.

Когда следует прекратить итерации? Если были установлены специфические требования практичности (UO, см. "Управление процессом разработки" далее), то итерации прекращаются, как только эти требования выполнены. В противном случае прекращение итераций – это управленческое решение, принимаемое на основе баланса стоимости и полезности дальнейших улучшений против необходимости выхода на рынок с законченным продуктом или сроков предоставления его пользователям.

2.9. Построение интерфейса системы

Ключевой руководящий принцип при построении (программной реализации) интерфейса состоит в обеспечении возможности его дальнейшего изменения. Постарайтесь предусмотреть некоторую настройку с помощью небольших изменений значений констант и переменных. Например, если вы пишете собственную подпрограмму для реализации специализированного меню, то не закладывайте жёстко в программу такие параметры, как размер, цвет, количество элементов. Постарайтесь также предусмотреть возможность небольших изменений кода, используя ясное разделение на модули. Если доработки в будущем потребуют замены специализированного меню какой-либо более общей функциональной возможностью, доработки кода должны быть тривиальны. Всё это звучит как обычные требования к хорошему стилю программирования и в действительности ими и является. Но это особенно важно для пользовательского интерфейса, который часто занимает более половины кода коммерческого продукта.

2.10. Отслеживание дизайна системы

Фундаментальный принцип рассматриваемого подхода состоит в том, что команда разработчиков не должна быть изолирована от всей остальной деятельности, связанной с функционированием системы. Если этот принцип уважается, то разработчик должен иметь контакт с пользователями не только в процессе разработки, но и после выпуска системы. Это ключевой момент для всех серьёзных организаций, заинтересованных в своём постоянном присутствии на рынке.

Один из методов введения разработчиков в контакт с пользователями – организация поочерёдных дежурств на горячей линии связи с потребителями. Другая важная вещь для больших систем – организация собраний групп пользователей и конференций. Такая работа также важна для менеджеров, т.к. позволяет им лучше понять реакцию пользователей на продукт, который они продают.

Кроме помощи в ответе на очевидный вопрос: делает ли система то, для чего была разработана, взаимодействие с пользователями часто рождает новые идеи об использовании продукта, открывая для него новые возможности на рынке. Эта информация может служить обратной связью для процесса разработки, позволяющей улучшить описание задач для новой версии продукта и углубить понимание разработчиком предметной области.

2.11. Изменение дизайна

На существующем сегодня рынке компьютеров и программ вряд ли существуют продукты, которые поддерживают свою жизнеспособность без регулярных усовершенствований (апгрейдов). Независимо от того, насколько удачно система была спроектирована первоначально, с большой вероятностью она будет терять адекватность с течением лет. Меняются и задачи и пользователи. Приёмы работы меняются из-за нового оборудования и программных продуктов. Пользователи приобретают новые навыки и ожидаемые реакции. Разработчики должны стоять вровень с этими изменениями, не только отслеживая состояние той рабочей среды, для которой была предназначена их система, но и развитие всего общества, технологий и методов, потребностей. Следующая версия системы должна не только устранять обнаруженные ошибки и недостатки, но и давать новые возможности пользователям.

2.12. Управление процессом разработки

Проблемно-центрированный подход против метода "водопада"

Традиционный метод "водопада" (waterfall model) начинает разработку с этапа анализа требований, который выполняется системными аналитиками, обычно не являющимися дизайнерами интерфейсов. Затем требования преобразуются в спецификации системы. Далее аппаратура, функциональная часть программного обеспечения и его интерфейсная часть разрабатываются в соответствии с принятыми спецификациями.

На практике оказалось, что метод водопада – плохой подход, если программное обеспечение содержит интерфейс с пользователем как важную компоненту. Как уже

говорилось, успешный разработчик интерфейса нуждается в глубоком понимании задач пользователя и того, как эти задачи вписаны в остальную работу пользователя. Такое понимание не может возникнуть из рассмотрения формальных спецификаций. Более того, опыт показывает, что для создания эффективного интерфейса существенны несколько итераций разработки. Традиционный метод водопада просто не допускает таких итераций.

Команда разработчиков

Так как методология проблемно-центрированного дизайна распространяет процесс разработки интерфейса на весь цикл разработки и жизненный цикл программного обеспечения, интерфейс не может быть сделан и проанализирован группой узких специалистов в одной точке всего процесса. Задача разработки хорошего интерфейса должна решаться командой, которая разрабатывает весь продукт в целом.

Команда разработчиков должна состоять из людей с разнообразными способностями и несколькими общими чертами. Они должны знать интересы пользователей, иметь опыт работы как с хорошими, так и с плохими интерфейсами, и, настроенные оптимистически, они должны быть привержены идее создания эффективной системы. Команда должна включать представителей всего диапазона интерфейсно-ориентированных специальностей: программистов, технических писателей, разработчиков учебных пакетов, специалистов по маркетингу. Команда может включать аналитика в области интерфейсов, но это не столь существенно. Разделяемая всеми приверженность качеству интерфейса, дополненная возможностями контактов с реальными пользователями, почти обязательно приведёт к хорошим результатам.

Ответственность

Ответственность за весь интерфейс должна быть централизована. В частности, разработчики, создающие программы, не должны от них отписываться и сдавать их совершенно отдельной группе, которая пишет руководства, которые затем сдаются другой группе, разрабатывающей обучающие пакеты. Вся эта деятельность должна координироваться, что достигается только посредством централизованного управления.

Требования практичности

Практичность (англоязычный термин – usability) – это одна из важнейших характеристик систем, изучению которой посвящено множество исследований. Управленческая деятельность в серьёзной корпорации может потребовать от вас представить различные показатели, которые количественно измеряют практичность. Требования практичности – это целевые значения для таких характеристик, как скорость выполнения репрезентативных задач и допустимое количество ошибок. Эти показатели могут использоваться, чтобы мотивировать разработчиков и обосновывать решения по распределению ресурсов. Целевые значения могут быть выбраны так, чтобы побить конкурентов или обеспечить функциональные нужды для хорошо определённых задач. Например, в целях успешной конкуренции от интерфейса может потребоваться не только полнота и удобство для пользователя, но и достижение результатов типа сокращения среднего времени выполнения задач пользователя на 20 % по сравнению с известными аналогами. Типичным примером специальной разработки в области интерфейса,

значительно улучшающим скорость работы, является алгоритм T9 для набора текстов на телефонной клавиатуре.