Лекция 4. Принципы создания удобного пользовательского интерфейса

Удобство использования программного обеспечения

Одним из важных показателей качества программного обеспечения является *удобство его использования.* Оно описывается с помощью таких характеристик, как понятность пользовательского интерфейса, легкость обучения работе с ним, трудоемкость решения определенных задач с его помощью, производительность работы пользователя с ПО, частота появления ошибок и жалоб на неудобства. Для построения действительно удобных программ нужен учет контекста их использования, психологии пользователей, необходимости помогать начинающим пользователям и предоставлять все нужное для работы опытных. Однако самым значимым фактором является то, помогает ли данная программа решать действительно значимые для пользователей задачи.

Многие программисты имеют технический или математический склад ума. Для таких людей «понятность», «легкость обучения» представляются весьма субъективными факторами. Сами они достаточно легко воспринимают сложные вещи, если те представлены в рамках непротиворечивой системы понятий, как бы дико эта система и входящие в нее понятия ни выглядели для постороннего наблюдателя. Такие люди чаще всего изучают новое ПО при помощи документации и искренне убеждены в том, что пользователи будут разбираться с написанной ими программой тем же способом. Типичный подход программиста при разработке пользовательского интерфейса — предоставить пользователю те же рычаги и кнопки, с помощью которых программист сам хотел бы управлять своей программой.

К пользователям, у которых возникли проблемы с программой, многие программисты достаточно суровы. Любимый их ответ в такой ситуации — «RTFM!» (read this fucking manual, прочти эту чертову инструкцию). Они любят рассказывать друг другу анекдоты о «ламерах», которые не в силах понять, что файлы нужно сохранять.

Посмотрим теперь, что будет с «обычным человеком», впервые попытавшимся воспользоваться компьютером вообще и текстовым редактором в частности (как ни тяжело представить такое сейчас).

Пользователь открывает редактор, скажем, Microsoft Word, как-то набирает текст, затем печатает его на принтере и выключает компьютер. Когда он включает компьютер в следующий раз и не находит важный документ, который он набрал (вы же сами помните!), он страшно раздосадован. Что вы говорите? Надо было сохранить документ? Что значит «сохранить»? Куда? Он же набрал документ и своими глазами видел, что «тот в компьютере есть». Зачем его еще как-то «сохранять»? Ну ладно, ну хорошо, раз вы так уверяете, что нужно нажать эту кнопочку, он будет ее нажимать. Да-да, каждые 10 минут, специально для вас, он будет нажимать эту кнопочку (зачем она нужна?...). Конечно же, спустя некоторое время он забудет это сделать.

Человек «понимает» смысл и назначение вещей и действий с ними, только если они в его сознании находятся в рамках некоторой *системы связанных друг с другом понятий.* Набор текста на компьютере больше всего напоминает набор текста на печатной машинке (и специально сделан выглядящим так в редакторах WYSIWYG), чуть менее он близок к письму. В

обоих этих случаях, написав или напечатав некоторый текст на листе бумаги, мы получим достаточно долго хранящийся документ. Чтобы избавиться от него, нужно предпринимать специальные действия — смять, порвать, пролить на него кофе, выкинуть в мусорную корзину. Если такой документ пропадает без наших действий, значит кто-то (сотрудник, начальник, жена, ребенок или уборщица) взял его. Человек, только что столкнувшийся с электронными документами, воспринимает их как аналоги бумажных и ожидает от них тех же свойств.

Документы «в компьютере» не такие. У компьютера есть два вида памяти — оперативная, или временная, и постоянная. В большинстве случаев набранный в редакторе текст находится в оперативной памяти, содержимое которой пропадает при отключении питания. Чтобы текст смог «пережить» это отключение, он должен быть перемещен в постоянную память. Именно для этого служит операция «сохранить документ».

В предыдущем абзаце описана некоторая система понятий, непривычная для новичка и не доступная с помощью непосредственного созерцания компьютера и размышлений. Ее необходимо как-то передать новому пользователю, иначе он не сможет понять, зачем же сохранять уже написанные документы, ведь они и так есть. Иначе, максимум, что он сможет сделать — выучить *ритуал,* согласно которому нужно иногда нажимать на кнопку «Сохранить». Очень многие люди работают с компьютерами и другой сложной техникой с помощью ритуалов, поскольку не всегда в силах разобраться в новой для них системе понятий, в рамках которой действует эта техника. Но гораздо чаще — потому, что ее производитель и разработчики не тратят столько усилий, сколько нужно, чтобы научить этой системе каждого пользователя.

Если же подпускать пользователей к компьютеру только после прочтения необходимой документации и усвоения ее информации, редко кто из них вообще заинтересуется использованием компьютера. Они используют компьютер и программное обеспечение только как инструменты для решения своих задач (единственный вид программ, где можно хоть как-то рассчитывать на чтение документации, — игры и развлечения), и им не хочется тратить время на чтение инструкций и осмысление правил, не относящихся напрямую к их области деятельности. Почему бы этим программам не быть столь же наглядными, как молоток и отвертка — никто не станет же всерьез читать инструкцию к отвертке?

Можно поспорить с этим на том основании, что компьютер и ПО намного сложнее отвертки, с их помощью можно выполнять гораздо больше действий и сами действия гораздо сложнее. Но, с другой стороны, умеют же сейчас делать автомобили, для вождения которых нужно знать только правила движения и основные элементы управления, а если что-то идет не так — пусть разбираются автослесари. Автомобиль сравним по сложности с самыми сложными программами, а то и превосходит их. И многие водители (по крайней мере, на Западе), используют автомобили, ничего не понимая в их устройстве. Пользователи изначально не хотят читать инструкции и не будут этого делать, пока эти инструкции занимают сотни страниц, написаны непонятным и сухим языком, требуют внимательности и обдумывания, не отвечают сразу на вопросы, которые интересуют пользователя в данный момент, а также пока начальство не скажет, что инструкцию все-таки прочитать надо. Но ведь есть еще и естественная человеческая забывчивость...

Удобство обычной, «не компьютерной» модели работы с документами подтверждается тем, что Palm Pilot, первый компьютер без разделения памяти на временную и постоянную, разошелся небывалым для такого устройства тиражом — за первые два года было продано около 2-х миллионов экземпляров.

Все сказанное выше служит иллюстрацией того факта, что «простота» и «легкость обучения» все-таки не совсем субъективны, а имеют объективные составляющие, которые необходимо учитывать при разработке части программного обеспечения, предназначенной для непосредственного взаимодействия с человеком — пользовательского интерфейса. Если посмотреть внимательнее, непонимание программистами пользователей в большой степени вызвано их, программистов, собственной ленью и нежеланием задумываться над непривычными вещами.

Рисунок 57. Что это**?** Лень или ошибка программиста**,** использовавшего не то стандартное окно **W**indows**?** А что делать пользователю**,** получившему такое сообщение**?**

Психологические и физиологические факторы

Вопросы удобства использования программного обеспечения тесно связаны с аналогичными вопросами для других видов инструментов и оборудования, а также предметов быта. И решаются они на примерно той же основе, что и вопросы типа «удобна ли эта дверная ручка», «удобно ли такое табло спидометра в автомобиле», «удобен ли данный способ управления станком» и пр.

На применяемые в этой области решения огромное влияние оказывают общие законы психологии и физиологии человека, ведь вещи удобны или неудобны большей частью не из-за субъективных предпочтений, а из-за того, что строение человеческого тела и законы работы сознания помогают или мешают использовать их эффективно.

Фундаментальной основой для определения удобств и неудобств понимания человеком функционирования и способов использования различных предметов является *когнитивная психология,* которая изучает любые познавательные процессы человеческого сознания. Психология использования машин, инструментов, оборудования и предметов обихода в ходе практической деятельности человека обычно называется *инженерной психологией*. За рубежом выделена особая наука, изучающая психологические, физиологические и анатомические аспекты взаимодействия человека и компьютера, которая так и называется — взаимодействие человека и компьютера (Human-Computer Interaction, HCI).

При рассмотрении задач построения удобного ПО используют много информации из

перечисленных дисциплин. Наиболее важные для разработки пользовательского интерфейса результаты этих дисциплин можно сформулировать следующим образом.

Человеку свойственно ошибаться

Обычный человек в нормальном состоянии постоянно делает ошибки разного рода. Можно сказать, что человек, в отличие от компьютера, является адаптивной аналоговой системой и успешность его «функционирования» в гораздо большей степени определяется не точностью выполнения действий и формулировки мыслей, а способностью быстро выдать хорошее приближение к нужному результату и достаточно быстро поправиться, если это необходимо.

Понаблюдайте за разговором двух хорошо знакомых людей — в нем очень часто встречаются обрывки фраз, восклицания и междометия. Далеко не каждая фраза досказывается и дослушивается до конца, и практически ни одна мысль не высказывается достаточно точно, чтобы быть однозначно понятной постороннему человеку, не включенному в контекст этого разговора. Очень часто высказываемая фраза и вовсе далека от выражаемой ею мысли, если на нее смотреть с точки зрения внешнего наблюдателя — просто большинство людей часто говорят не то, что хотели бы сказать, а то, что у них быстрее получается. Тем не менее, они понимают друг друга, а послушав их некоторое время, и третий человек начинает понимать, о чем идет речь.

Поэтому один из принципов построения удобных систем — терпимость к человеческим ошибкам, умение не замечать их, «понимая» пользователя правильно, несмотря на его не вполне корректные действия, а также наличие возможностей полного устранения последствий совсем уж неверных действий. Многими специалистами по удобству использования достаточно серьезно воспринимается радикальный тезис, состоящий в том, что пользователи не ошибаются, а лишь выполняют «действия, не направленные на достижение своих собственных целей».

К тому же сообщения об ошибках, которыми программы пугают неопытных пользователей, являются чаще всего отвлекающим от работы фактором, приводят к раздражению, а иногда — к отказу от работы с программой, которая «слишком умничает». Именно так пользовательвоспринимает указания программы, которая, явно не понимая, что же хочет человек, пытается заявлять о неверности его действий. Сообщений об ошибках в удобной программе практически не должно быть, а те, которые все-таки не удается убрать, ни в коем случае не должны формулироваться недостаточно информативно и категоричным тоном: «Неправильно! Некорректное значение!», как будто пользователь сдает экзамен. Человеку будет гораздо комфортнее, если система признается, что не может понять, что он хочет, объяснит, какие именно из введенных им данных вызывают проблемы и почему, а также предложит возможные варианты выхода из создавшейся ситуации. Но еще лучше — если ПО все понимает «правильно», даже если пользователь ошибся, и не обращает внимания на подобные мелочи.

Рисунок 58. Почему **11,3 —** неправильное число**?**

С другой стороны, действия, связанные с большим риском и невозможностью отмены, не должны быть легко доступны — иначе простота доступа в сочетании с человеческими ошибками будет часто приводить к крайне нежелательным последствиям. В самих программах таких действий немного — лишь потеря большого количества ценных данных может претендовать на такой статус, но ПО может управлять и кораблем, и самолетом, и атомной электростанцией. В подобных случаях необходимо особо выделять действия, выполнение которых может привести к тяжким последствиям. Не стоит делать запуск ракеты сравнимым по простоте с удалением файла.

Скоростные показатели деятельности человека

Время, которое человек затрачивает на различные действия, связанные с работой на компьютере, таково.

- Нажатие на клавишу клавиатуры: 0.2-1.25 с.
- Нажатие на кнопку мыши: 0.1 с.
- Перемещение курсора мыши (см. ниже о законе Фиттса): 1.0-1.5 с.
- Значительное время затрачивается и на действия, которые тоже постоянно необходимо выполнять, хотя они не имеют такого прямого физического выражения, как предыдущие.
- Распознавание визуального образа: 0.1 с.
- Перевод взгляда и переключение внимания с одного объекта на другой: 0.25 с.
- Выбор из двух альтернатив (принятие простейшего решения): 1.25 с.
- Переключение внимания с мыши на клавиатуру и обратно: 0.36 с.

На данных такого рода основан GOMS — метод оценки производительности работы с интерфейсом (см. далее).

Наблюдения показывают, что большую часть времени при работе человек тратит на интеллектуальную деятельность, связанную с определением целей своих действий, определением цепочки конкретных действий, которую нужно совершить, чтобы достичь поставленных целей, обнаружением всех необходимых для этого элементов, распознаванием очередного состояния системы и его оценкой с точки зрения достижения выбранных целей. Изменяя пользовательский интерфейс, можно лишь помочь пользователю быстрее найти

нужные ему инструменты, выполнить нужные действия и быстрее понять, какие же результаты они дали.

Из приведенных данных можно сделать несколько важных выводов.

- Человек воспринимает и осознает информацию, а также производит действия достаточно медленно по сравнению с компьютером. Сама «медленность» действий человека и его восприятия, а также соотношения затрат времени на различные действия должны учитываться при проектировании интерфейсов, рассчитанных на взаимодействие с человеком.
- Глаз быстрее руки — человек гораздо быстрее узнает что-то, чем производит соответствующие действия. Поэтому, в частности, человеку часто удобнее работать с системами контекстной подсказки, предлагающими ему возможные варианты его дальнейшего ввода, чем набивать весь текст целиком самостоятельно.

В качестве отдельного наблюдения можно заметить, что человек гораздо быстрее узнает что-то, чем вспоминает, как оно называется. Значительно проще указать мало знакомого человека на фотографии, чем вспомнить его имя и фамилию. Точно так же проще выбрать какой-то элемент из предоставленного списка, чем набрать его идентификатор или имя.

Приведенные значения для времени перемещения указателя мыши можно уточнить с помощью закона Фиттса (Fitts) $T = a + b \cdot \log(D/W)$ (иногда используется формула $a + b \cdot (D/W)^{1/2}$ Здесь D обозначает расстояние, на которое переносится указатель, W — линейный размер объекта, a и b — некоторые константы, зависящие от человека и устройства мыши. Этот закон говорит, что чем ближе и больше элемент управления, тем быстрее можно попасть в него с помощью мыши.

Из этого следует, что удобнее располагать нужные пользователю элементы управления ближе к указателю мыши и делать их крупнее. Кроме того, поскольку мышь нельзя вывести за край экрана, элемент, расположенный на краю, воспринимается как «бесконечно большой» — попасть в него гораздо легче, чем в элемент аналогичного размера и на том же расстоянии от указателя, но со всех сторон окруженный «пустым» пространством.

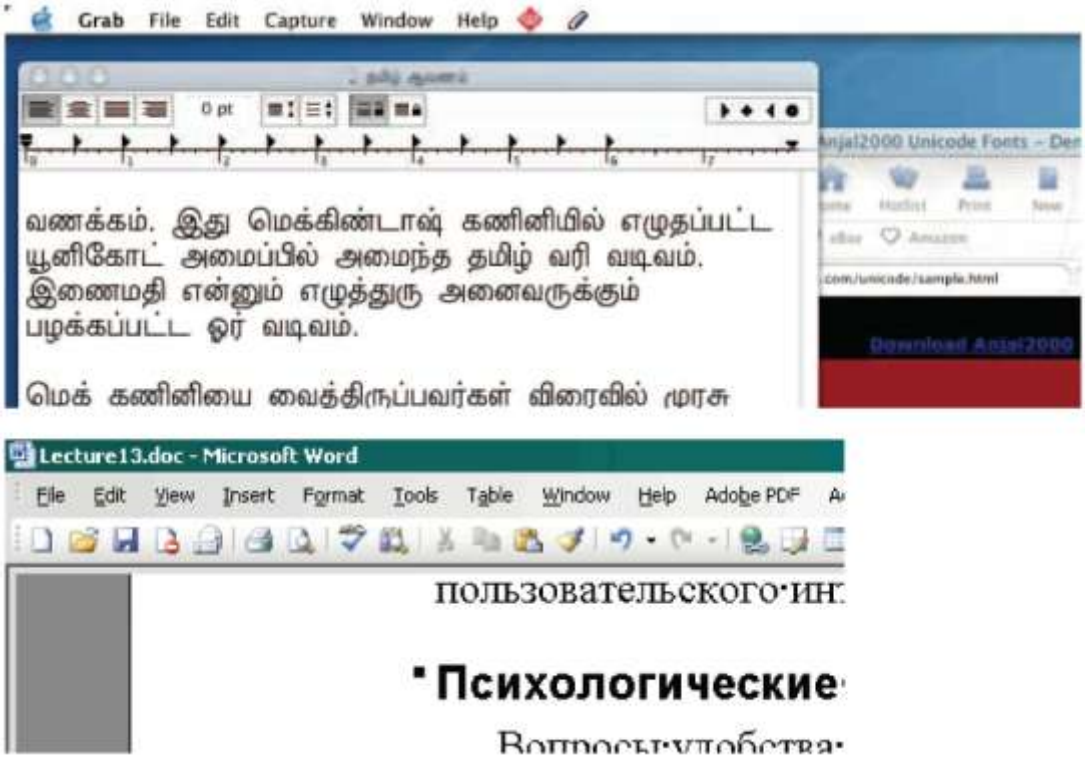

Рисунок 59**.** Добираться до меню в **MacOS (**сверху**)** несколько удобнее**,** чем в **W**indows **(**снизу**),** поскольку в первом случае оно расположено вдоль края экрана**.** Правда**,** надо привыкнуть к тому**,** что меню отделено от окна программы**.**

Люди чаще всего промахиваются при первой попытке попасть указателем в нужное место, но большинство быстро поправляется, даже не осознавая произошедшего промаха. Если система снисходительна к промахам указателя, которые быстро исправляются, она удобнее той, которая немедленно реагирует на такие события. Например, панель задач в Windows выпадает сразу по наведении на нее указателя мыши, поэтому при попытках нажать кнопку, расположенную где-то внизу экрана, она часто появляется «неожиданно», и пользователю приходится терять время на ожидание того, чтобы она скрылась обратно.

Внимание человека

Человеку очень тяжело долго сохранять сосредоточенность и не отвлекаться от какой-то деятельности. Чаще всего, помимо воли человека, через некоторое время ему в голову приходят разные мысли, и взгляд сам собой уползает куда-то в сторону.

Это означает, что нельзя требовать от человека длительного внимания к работе. Само понимание того, что необходимо сосредоточиться, создает у него стресс и вызывает негативные эмоции. Только в самых рискованных ситуациях, когда от его действий зависит очень многое — пике самолета, превышающий все нормы разогрев химического реактора — пользователь может сосредоточиться на значительное время, но все равно мы все предпочитаем избегать подобных обстоятельств. Такие ситуации должны явно отмечаться какими-то характерными и хорошо ощущаемыми сигналами (сирена, красный мигающий свет и пр.), и ни в коем случае эти же или похожие сигналы нельзя использовать для гораздо менее серьезных случаев, иначе пользователи будут сбиты с толку и могут их перепутать, что приводит к печальным последствиям.

В остальных же ситуациях ПО должно быть готово к постоянным переключениям внимания пользователя и ненавязчиво помогать ему восстановить фокус внимания на последних действиях, которые он выполнял, а также вспомнить их контекст — что он вообще делал, на каком шаге он сейчас находится, что еще нужно сделать и пр.

Поэтому, например, экранные формы, заполнение которых входит в одну процедуру, хорошо бы помечать так, чтобы пользователь сразу видел, сколько шагов он уже выполнил и сколько еще осталось. А текущее поле ввода стоит как-то выделять среди всех полей ввода на форме.

Для привлечения внимания пользователей к каким-то сообщениям или элементам управления нужно помнить правило: *сначала движение, затем яркий цвет, затем все остальное.* Внимание человека прежде всего акцентируется на движущихся объектах, потом — на выделенных цветом, и только потом на остальных формах выделения. Лучшим способом привлечения внимания является появление анимации, чуть менее действенным — яркие цветные окна и сообщения. Для мягкого, не режущего глаз привлечения внимания можно использовать выделение при помощи цветов мягких оттенков или изменения шрифта сообщений.

Наоборот, появление на экране или страничке ненужных цветных анимированных картинок является лучшим способом отвлечь человека от выполняемой им работы: глаза поневоле переводятся на такую картинку, и требуется психологическое усилие и некоторое время, чтобы от нее оторваться. Некоторые люди даже закрывают анимацию рукой, чтобы она не мешала сосредоточиться на полезной информации на экране!

Ярко выделенных элементов на одном экране не должно быть много, не больше, чем один-два, иначе человек перестает обращать внимание на такое выделение.

Другой фактор, связанный с вниманием пользователей, — их оценка времени выполнения системой заданных действий. Если человек ожидает, что система будет печатать документ достаточно долго, то, послав его на печать, он пойдет налить себе чаю и вернется к тому времени, когда, по его оценке, система должна закончить работу. Поэтому, прежде чем выполнять долгие операции, нужно убедиться, что все необходимые данные от пользователя получены. Иначе и пользователь, и система будут терять время, первый — ожидая на кухне, когда же система выполнит работу (которую он вроде бы запустил), а вторая — ожидая ввода дополнительных данных.

По той же причине стоит аккуратнее относиться к индикаторам степени выполнения, поскольку по их показаниям пользователи часто оценивают оставшееся время до конца выполнения задачи.

Понятность

Человек «понимает» что-либо, укладывая это в сознании в некоторую систему ассоциативных связей. Примерами механизмов, которые помогают в этом, являются следующие.

1. *Ментальная модель.* Этот механизм иллюстрируется примером с сохранением файлов, окотором рассказывалось выше. Пользователь понимает, как работать с системой, если емуобъясняют набор сущностей, в терминах которых она функционирует, и правила работы с ними.

Человек, один раз понявший модель действий какой-то системы, может помнить ее достаточно долго и воспринимает большинство происходящих в системе событий как вполне естественные, даже если управляющие ею правила достаточно сложны и неочевидны.

Проблема в том, что модели работы многих систем сейчас довольно сложны, а большинство людей с трудом воспринимают сложные модели, и, начиная с некоторого уровня сложности, — не воспринимают вообще. К тому же для обучения людей такой модели нужно тратить дополнительные усилия, или же нужны дополнительные стимулы, чтобы люди прочитали документацию, в которой она объясняется.

2. *Метафора.* Сейчас довольно часто объяснения того, как работает ПО, строятся вокруг той или иной метафоры, т.е. приводится какой-то механизм или предмет из реальной жизни (или литературы), знакомый большинству пользователей, и говорится, что данное ПО работает так же. Это помогает понять основные правила его работы очень быстро, даже если их достаточно много, в отличие от долгих объяснений модели.

Примером метафоры служит рабочий стол Windows. Для описания его работы привлекается аналогия с рабочим столом офисного служащего, на котором лежат различные документы и инструменты. Этот же пример достаточно наглядно показывает границы применимости метафоры для повышения понятности интерфейса — вряд ли с помощью этой метафоры можно распространить понимание работы Windows за пределы работы с файлами и папками, лежащими на рабочем столе.

К сожалению, очень редко имеется или может быть придумана достаточно адекватная метафора для значительной части системы. Еще более редко такая метафора может охватить систему целиком. Иначе правила работы ПО покрываются набором слабо связанных метафор, и нужно постоянно помнить границы их применимости, а пользователи могут запутаться в этом нагромождении. Но даже в таком случае метафоры помогают, если четко ограничить зоны их действия.

3. *Наглядность (affordance).* Свойство наглядности означает, что само представление интерфейса подсказывает, как с ним надо работать, — оно использует широко распространенные стереотипы и наглядные (не обязательно видимые глазами!) связи между элементами управления и управляемыми объектами.

Примеры наглядности в ПО и других областях:

a . Псевдотрехмерные выступающие кнопки как бы предлагают «Нажми меня!». При нажатии они на время «утапливаются», закрепляя таким образом наглядную ассоциативную связь с обычными кнопками.

b. Если расположить вентили для управления конфорками газовой плиты в виде той жегеометрической фигуры, как и сами конфорки (обычно по углам квадрата), соответствие между вентилями и управляемыми ими конфорками становится гораздо яснее, чем при традиционном расположении вентилей на одной прямой.

Придумать интерфейс, имеющий свойство наглядности, очень непросто, однако если это удается, такая система почти всегда становится крупным событием на рынке.

Гораздо чаще, к сожалению, можно найти примеры *антинаглядности* — интерфейс всем

своим видом «говорит» одно, а использовать его надо совсем иначе, скажем, нарисована кнопка, а нажать ее нельзя. Антинаглядности необходимо избегать всеми силами. Например, если дверная ручка, «предлагающая» повернуть себя (ручка, за которую удобно хвататься, с одним свободным концом, а другим закрепленная на круглой ножке), не поворачивается, чаще всего ее нечаянно отламывают.

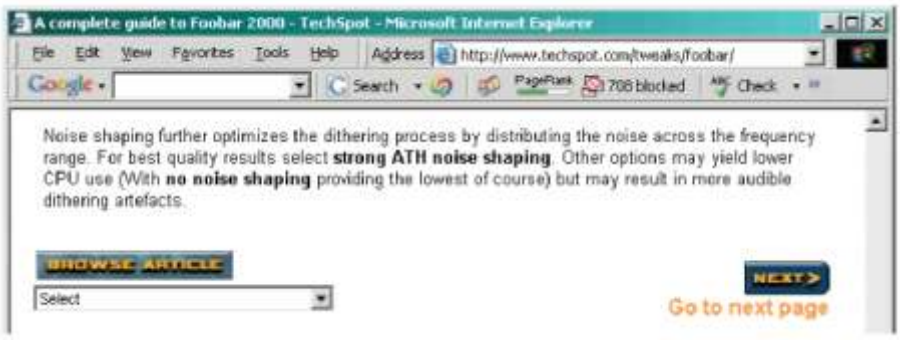

Рисунок 60. Антинаглядность. "Кнопка" Next не нажимается

4. *Стандарт.* Человек хорошо воспринимает то, что привычно. Одним из способов «внедрения» нужных привычек является стандартизация и широкое использование стандартных интерфейсов. В последнее время стандартам внешнего вида интерфейса в целом и его отдельных элементов придается большое значение в плане обеспечения удобства использования. Этот подход действительно уменьшает затраты на обучение работе с новой программой, поскольку пользователям становится легче ориентироваться в ней. Иногда удается выработать хороший стандарт, который значительно облегчает жизнь и пользователям, и разработчикам.

Однако значительно повысить удобство использования, особенно для опытных пользователей, только за счет применения стандартных элементов интерфейса чаще всего не удается. Это происходит потому, что стандарт, во-первых, закрепляет не все необходимые детали, во-вторых, предоставляет далеко не самые удобные, а иногда и совсем неудобные элементы. Например, стандартным средством для работы с документами, которые не вмещаются в рамки окна программы, на некоторое время стали полосы прокрутки, реализующие неудобный для человека способ перемещения по большому пространству. Сопоставьте свои действия и движения при разглядывании большой карты или схемы с тем, что приходится предпринимать при использовании для этого полос прокрутки.

Память человека

Обычно человеческую память разделяют на кратковременную и долговременную. Широко известно, что в кратковременную память помещается не более 5-9 (7±2) объектов. Долговременная же память хранит информацию в виде некоторой структуры с большим количеством связей между элементами; чтобы поместить туда нечто, нужно связать эту вещь с несколькими другими, лучше уже знакомыми.

Отсюда часто делают вывод, что 7 элементов на окне диалога — это нормально, а 10 — уже плохо. Это не совсем так. С одной стороны, довольно много людей имеет «объем» кратковременной памяти, равный 5 или 6 — для них 7 элементов уже много. С другой стороны,

границы между двумя видами памяти менее резкие, чем это обычно представляется. Ограничения на число элементов в кратковременной памяти касаются несвязанных по смыслу вещей — последовательность букв ОДТЧПШСВДН запомнить с первого раза тяжело, но если знать, что это — первые буквы русских названий цифр, заканчивающихся нулем, буквы очень легко восстанавливаются. При этом используются связи между вещами (буквами и словами, самими словами и их смыслом), что более характерно для долговременной памяти.

Из сказанного можно заключить, что на формах лучше делать все же не 7, а 5, максимум 6 элементов, не связанных друг с другом, а для расширения возможностей кратковременной памяти нужно привлекать смысловые (или ассоциативные) связи между ними, например, разбить элементы, которых больше 5-ти, на меньшее число смысловых групп. Использование таких связей оказывается работоспособным механизмом и при обучении, во время которого пользователь должен запомнить довольно много.

В то же время, чем меньше нагружается память пользователей, тем меньше усилий им необходимо для работы с ПО. Предоставление нужной информации где-то на экране способно значительно облегчить им жизнь. И уж точно плохи программы, при работе с которыми пользователю приходится иногда записывать что-то на бумажке, чтобы затем использовать это в другой экранной форме или другом окне — посторонний наблюдатель сразу выразит недоумение таким способом работы, хотя сами пользователи часто привыкают к подобным вещам.

Стоит иметь в виду и постепенность запоминания — на изучение всего нового требуется время. Пользователи, не имеющие опыта работы с программой, поначалу совершают довольно много ошибок, даже пытаясь выполнять уже знакомые операции. Постепенно они вырабатывают некоторые шаблоны действий и уже не делают столько ошибок. Тем не менее, на начальном этапе им может быть необходима дополнительная помощь, чтобы не потерять интерес к изучению программы.

Разные категории пользователей

Необходимо обращать внимание на возрастные и другие особенности людей. Использование в интерфейсе программы мелких шрифтов, маленьких кнопок и пиктограмм способно сделать ее недоступной для тех, кому за 40. Использование только цветов для различения каких-то элементов интерфейса сделает его неудобным для людей с нарушениями цветового восприятия (а это около 10% всех возможных пользователей).

Конечно, если ваша программа предназначена для использования только молодежью или в организации, где могут работать только люди с нормальным цветовым зрением, можно не обращать на это внимание. Если же контекст использования программы позволяет людям с ограниченными возможностями пользоваться ею, стоит аккуратно оценивать ее удобство с их точки зрения и не пренебрегать теми элементами, которые могут им помочь.

Например, при настройках по умолчанию различные графические элементы Windows окрашены в небольшое число цветов, используются только оттенки синего и серого цветов. Это сделано для того, чтобы люди с наиболее часто встречающимися нарушениями цветового восприятия могли уверенно работать с системой. Нарушения цветового зрения в синей части

спектра встречаются реже всех остальных.

Если же программа предназначена, наоборот, только для использования особой категорией людей, например, для обучения умственно отсталых детей, необходимо тщательно исследовать особенности их восприятия и разрабатывать интерфейс целиком для их удобства.

Факторы удобства использования и принципы создания удобного ПО

Основные факторы, с помощью которых можно оценить или даже измерить удобство использования программы, следующие.

- ***Адекватность интерфейса.***

Адекватность пользовательского интерфейса программы — это его соответствие тем задачам, которые пользователи должны и хотели бы решать с ее помощью. Это соответствие имеет два аспекта: во-первых, все нужные пользователям задачи должны быть разрешимы (если это не так — у программы большие проблемы), во-вторых, те действия, которые пользователи выполняют чаще, должны требовать меньше усилий.

- ***Производительность работы пользователей.***

Это количество однотипных реальных задач, которые пользователь может решить с помощью ПО за единицу времени.

- ***Скорость обучения новых пользователей.***

Это количество задач, выполнению которых новый пользователь самостоятельно обучается за единицу времени.

Кроме того, важным показателем является соответствие обучения частоте возникновения задач — чем чаще на практике возникает необходимость решить определенную задачу, тем быстрее пользователь должен научиться делать это.

- ***Эффективность предотвращения и преодоления ошибок пользователей.***

Этот показатель тем лучше, чем реже пользователи ошибаются при работе с данным интерфейсом и чем меньше времени и усилий требуется для преодоления последствий уже сделанных ошибок.

Стоит особо отметить, что предотвращение ошибок, в том числе «правильное» понимание программой действий пользователя, не вполне совпадающих с теми, которые в данном контексте хотели бы видеть ее разработчики, гораздо важнее. Предотвращенная ошибка — это ошибка не сделанная, пользователь не считает ее ошибкой и не испытывает никакого стресса по ее поводу, она не снижает его производительность.

Большое значение имеет также риск, связанный с возникновением ошибки. Удобное ПО не должно требовать от пользователя серьезных усилий на совершение действий, ошибки в которых не слишком накладны; но если последствия ошибки катастрофичны, необходимо всячески препятствовать ее совершению. Именно поэтому запуск боевой ракеты или открытие хранилища банка не выполняются нажатием одной кнопки, которая всегда под рукой, а обставлены трудностями, вроде необходимости сделать два-три существенно разных действия, одновременно повернуть два ключа, и пр.

• ***Субъективное удовлетворение пользователей.***

Этот фактор наиболее тяжело проанализировать, хотя измерить его довольно просто — нужно попросить группу пользователей оценить, понравилась им система или нет, по какой-то шкале (обычно используются 5-ти или 7-ми-бальные шкалы). Средний результат полученных оценок можно использовать как числовое значение этого показателя. Тяжелее *добиться,* чтобы интерфейс хорошо воспринимался пользователями в среднем — простое внесение исправлений, подсказанных одними пользователями, может отрицательно сказаться на оценках других. Считается, что субъективное удовлетворение пользователей почти всегда повышается в следующих случаях:

- Если интерфейс программы эстетичен и элегантен, т.е. построен на немногих и близких к гармоничному (1:1.618) соотношениях между размерами отдельных элементов и расстояниями между ними, на мягких, неброских цветах и не режущих глаз их сочетаниях, небольшом количестве контрастов, слегка сглаженных углах, на аккуратном выравнивании отдельных элементов.
- Если в работе системы не возникает долгих пауз, во время которых пользователи не знают, чем заняться. Даже если системе требуется много времени для выполнения каких-то действий, показываемые в это время картинки и предоставляемая дополнительная информация могут снизить субъективную длительность ожидания.
- Если у пользователей, даже достаточно неопытных, не возникает стресса при работе с системой. Стресс может возникнуть по многим причинам:
  - От осознания ответственности за производимые действия и невозможности отменить их, если они вдруг окажутся неправильными.
  - От необходимости поддерживать высокий уровень внимания, запоминать какие-то вещи, чтобы использовать их позднее, или серьезно обдумывать выполнение каждого очередного действия.
  - От отсутствия контроля за поведением системы, когда она выполняет непонятные действия или сообщает о событиях, которые сам пользователь не инициировал (или не может связать их со своими предыдущими действиями). Другой источник потери ощущения контроля — изменчивость самого интерфейса. Если кнопки, элементы меню и пр. то появляются в определенных местах, то исчезают или становятся неактивными по неизвестным (ненаблюдаемым непосредственно) причинам, пользователь теряется. Связанный с таким поведением системы стресс объясняется также невозможностью быстро построить в сознании модель ее поведения — система становится чем-то зыбким, неустойчивым.
  - От частых, категоричных и неинформативных сообщениях об ошибках.

- Удовлетворение пользователей выше, если дается возможность настроить интерфейс программы так, как им нравится, но при этом предоставленных вариантов не чересчур много, чтобы пользователь не мог в них «заблудиться».

Специалисты по удобству использования обычно формулируют некоторый набор принципов и правил, позволяющих как оценивать удобство интерфейса, так и предлагать решения, повышающие его удобство. Ниже приведены такие правила.

- **Правило *доступности*.**

Система должна быть настолько понятной, чтобы пользователь, никогда раньше не видевший ее, но хорошо разбирающийся в предметной области, мог без всякого обучения начать ее использовать.

Это правило служит некоторым идеалом, к которому надо стремиться, поскольку на практике достичь такой степени понятности почти никогда не удается. Тем не менее, все, что можно сделать для достижения этого идеала, делать нужно.

- ***Правило* эффективности.**

Система не должна препятствовать эффективной работе опытных пользователей, работающих с ней долгое время.

Очевидным примером нарушения этого правила является нацеленность системы только на новичков, например, выполнение почти всех операций с помощью мастеров (wizards), которые хорошо подходят для неопытного пользователя, ограничивая его в возможности сделать что-то не так, но неэффективны для эксперта, который и так знает, что и где ему нужно сделать.

- **Правило непрерывного развития.**

Система должна способствовать непрерывному росту знаний, умений и навыков пользователя и приспосабливаться к его меняющемуся опыту. Плохие результаты приносит предоставление только базовых возможностей или оставление начинающего пользователя наедине со сложным интерфейсом, которым уверенно пользуются эксперты. Нарушение непрерывности при переходе от одного набора возможностей к другому также приносит неудобства, поскольку пользователь вынужден разбираться с добавленными возможностями в новом контексте.

Большинство пользователей можно поместить в три группы: новичков, опытных и средних, которые уже знают больше, чем новички, и не делают столько ошибок, но еще не приобрели автоматизма при выполнении большинства операций и иногда путаются в интерфейсе. Новичкам необходима помощь в освоении новой для них системы и контроль их действий, опытным пользователям — высокая эффективность выполнения часто требующихся действий и возможность гибкого управления системой в таких ситуациях, которые встречаются реже, но способны вызвать проблемы при их неадекватной поддержке. Про средних же пользователей часто забывают, хотя подавляющее большинство пользователей ПО относится к этой категории. Им нужны достаточно высокие эффективность и гибкость вместе с возможностью быстро получать адекватную помощь по возникающим время от времени разнообразным вопросам.

- **Правило поддержки.**

Система должна способствовать более простому и быстрому решению задач пользователя. Это означает, прежде всего, что система должна *действительно решать* задачи пользователя. Кроме того, она должна решать их *лучше, проще и быстрее,* чем имевшиеся до ее появления инструменты и методы.

- **Правило соблюдения контекста.**

Система должна быть согласована с контекстом, в котором ей предстоит работать. Это правило требует от системы быть работоспособной не «вообще», а именно в том окружении, в котором ею будут пользоваться. В контекст могут входить специфика и объемы входных и выходных данных, тип и цели организаций, в которых система должна работать, уровень пользователей, зашумленность помещений и пр.

Представленные выше правила определяют общие требования, которым должен удовлетворять удобный интерфейс. Кроме них при разработке ПО необходимы некоторые подсказки, позволяющие сделать его более удобным.

Следующие *принципы* позволяют находить решения, повышающие удобство пользовательского интерфейса.

- **Принцип структуризации.**

Пользовательский интерфейс должен быть целесообразно структурирован. Близкие по смыслу, родственные его части должны быть связаны видимым образом, а независимые — разделены; похожие элементы должны выглядеть похоже, а непохожие — различаться.

- **Принцип простоты.**

Наиболее распространенные операции должны выполняться максимально просто. При этом должны быть видимые ссылки на более сложные процедуры.

- **Принцип видимости.**

Все функции и данные, необходимые для решения определенной задачи, должны быть видны, когда пользователь пытается ее решить.

- **Принцип обратной связи.**

Пользователь должен получать сообщения о действиях системы и о важных событиях внутри нее. Сообщения должны быть информативными, краткими, однозначными и написанными на языке, понятном пользователю.

- **Принцип толерантности.**

Интерфейс должен быть гибким и терпимым к ошибкам пользователя. Ущерб от ошибок должен снижаться за счет возможности отмены и повтора действий и за счет разумной интерпретации любых разумных действий пользователя и введенных им данных. По возможности, следует избегать обязывающего взаимодействия (модальных диалогов), основанного на ограничении свободы пользователя.

- **Принцип повторного использования.**

Следует стараться использовать многократно внутренние и внешние компоненты, обеспечивая тем самым унифицированность интерфейса и сходство между похожими его

элементами.

Методы разработки удобного программного обеспечения

Одним из наиболее технологичных подходов к разработке удобного пользовательского интерфейса является *проектирование, ориентированное на использование* (usage-centered design), предложенное Л. Константайном и Л. Локвуд (L. Constantine, L. Lockwood) [3].

Основная идея этого метода — использование специальных моделей, способствующих адекватному определению набора задач, которые необходимо решать пользователям, и способов организации информации, позволяющих упростить их решение.

Список моделей, которые используются в рамках проектирования, ориентированного на использование, следующий.

- **Модель ролей.**

Эта модель представляет собой список ролей пользователей системы. Каждая *роль* — это группа связанных задач и потребностей некоторого множества пользователей. Модель ролей может определять связи между ролями (роли могут уточнять друг друга, включать друг друга или просто быть похожими) и набор из одной-трех *центральных ролей,* на которые, в основном, и будет нацелено проектирование.

Кроме того, каждая роль может быть снабжена профилями, указывающими различные ее характеристики по отношению к контексту использования системы. Профили могут включать следующую информацию:

- Обязанности — требования к знаниям (о предметной области, о самой системе и пр.), которым пользователь в данной роли, скорее всего, удовлетворяет.
- Умения — уровень мастерства в работе с системой.
- Взаимодействия — типичные варианты взаимодействия пользователя в этой роли с системой, включая их частоту, регулярность, непрерывность, концентрацию, интенсивность, сложность, предсказуемость, а также управление взаимодействием (направляется ли оно пользователем, или он только реагирует на действия системы).
- Информация — источники, объем, направление передачи и сложность информации при взаимодействии с системой
- Критерии удобства — специфические критерии удобства работы для данной роли (быстрота реакции, точность указаний, удобство навигации и пр.).
- Функции — специфические функции, возможности и свойства системы, необходимые или полезные для данной роли.
- Возможные убытки от ошибок, которые может совершить человек в данной роли, риски использования различных функций.

Пример модели ролей для пользователей банкомата приведен на Рис. 61.

Рисунок 61. Модель ролей пользователей банкомата

- **Модель задач.**

Модель задач при проектировании пользовательского интерфейса строится на основе *сущностных вариантов использования (essential use cases).* Описание сущностного варианта использования отличается от обычного тем, что в рамках его сценариев выделяются только цели и задачи пользователя, а не конкретные его действия.

Действия	**Реакции**
Вставить карту	
	Считать данные
	Запросить PIN
Ввести PIN	
	Проверить PIN
	Вывести меню
Нажать клавишу выдачи денег	
	Запросить сумму
Ввести сумму	
	Вывести сумму
	Запросить подтверждение
Нажать клавишу подтверждения	
	Вернуть карту
	Выдать деньги
	Напечатать чек
	Выдать чек

Задачи	**Обязательства**
Регистрация в системе	
	Проверка личности
	Предложение набора операций
Выбор операции выдачи денег	
	Выдача денег
	Выдача чека

Таблица 9. Описание обычного **(**слева**)** и сущностного **(**справа**)** вариантов использования**.**

Целью такого выделения является освобождение от неявных предположений о наличии определенных элементов интерфейсов, что помогает разрабатывать их именно для решаемых

задач. Удобно описывать такие сценарии в виде двух последовательностей — устремлений пользователя (не действий, а задач которые он хочет решить) и обязательств системы в ответ на эти устремления.

Пример описания обычного и сущностного варианта использования при работе приведен в Таблице 9.

В результате модель задач представляет собой набор переработанных вариантов использования со связями между ними по обобщению, расширению и использованию. Некоторые из вариантов использования объявляются основными — без них программа потеряет значительное количество пользователей.

Всякая пользовательская роль при этом должна быть связана с одним или несколькими вариантами использования.

- **Модель содержимого.**

Модель содержимого пользовательского интерфейса описывает набор взаимосвязанных *контекстов взаимодействия или рабочих пространств* (представляемых экранами, формами, окнами, диалогами, страницами и пр.) с содержащимися в них данными и возможными в их рамках действиями.При построении этой модели нужно определить, *что* войдет в состав интерфейса (какие данные и функции), и не решать вопрос о том, *как именно* оно будет выглядеть. На начальном этапе один контекст взаимодействия ставится в соответствие одному (не вспомогательному!) варианту использования или группе очень похожих вариантов, для выполнения которых понадобится один и тот же набор инструментов.

Средства для поддержки вспомогательных расширяющих вариантов использования обычно удобно помещать в контексты расширяемых ими основных вариантов.

Сначала устанавливается, какая информация должна находиться в заданном контексте дляуспешного решения задач соответствующего варианта использования, затем определяетсясписок необходимых операций для работы с этой информацией.

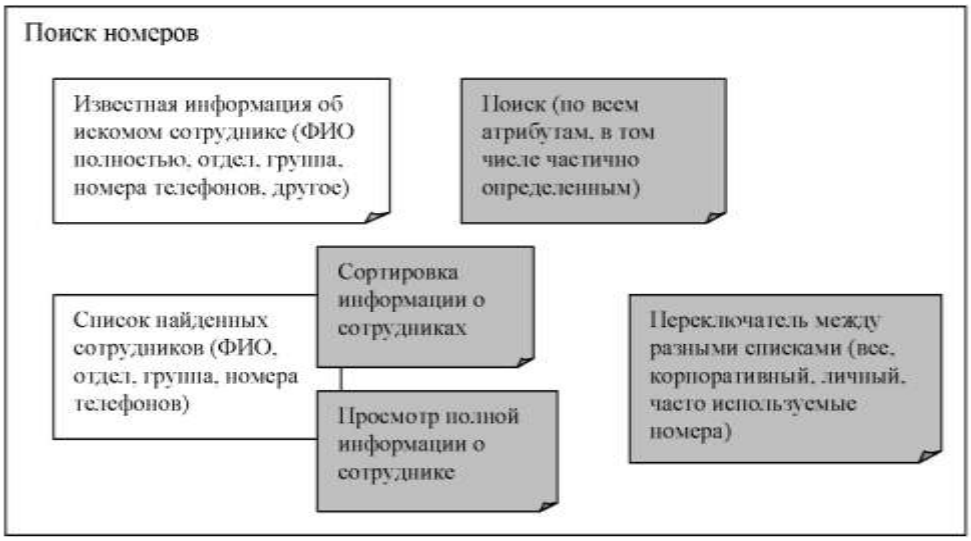

Рисунок 62. Пример модели содержимого контекста взаимодействия**.**

Часто при обсуждении содержимого контекста взаимодействия для его представления

используют лист бумаги с наклеенными на него подписанными стикерами разных цветов (для различения информационных элементов и элементов управления). Такое представление удобно для быстрого внесения изменений по ходу обсуждения. Оно также наглядно показывает, что рассматривается лишь прототип окна или странички, а его элементы абстрактны, для них пока не предлагается конкретная форма. Его трудно принять за «почти готовый» проект окна, формы или странички, описывающий итоговые форму, расположение и цвета элементов интерфейса.

На рис. 62 приведен пример модели содержимого окна поиска номеров телефонов программы, реализующей корпоративный телефонный справочник.

После определения набора контекстов и их информационного и функционального содержимого рисуется *карта навигации* между контекстами, показывающая возможные переходы между ними. Карта навигации объединяет различные контексты взаимодействия в рамках модели содержимого интерфейса.

Рисунок 63. Часть карты навигации редактора MICROSOFT WORD.

После разработки модели содержимого всякий основной вариант использования должен быть поддержан при помощи одного или нескольких контекстов взаимодействия. Чем меньше контекстов нужно использовать для выполнения одного варианта использования, тем лучше.

Перечисленные три вида моделей — ролей, задач и содержимого — являются основными. Оставшиеся два вида моделей используются только при необходимости.

- **Операционная модель** описывает контекст использования системы и состоит из профилей пользовательских ролей.
- **Модель реализации** представляет собой визуальный проект интерфейса и описание его работы.

Основные виды деятельности в рамках проектирования, ориентированного на

использование, следующие:

- Совместное с пользователями определение требований к ПО, с учетом пожеланий и требований к его интерфейсу.
- Разработка модели предметной области с помощью пользователей.
- Разработка моделей ролей и задач с помощью пользователей.
- Разработка модели содержимого.
- Разработка визуального проекта интерфейса (модели реализации).
- Контроль удобства использования проекта интерфейса с участием пользователей.
- Проектирование объектной структуры ПО.
- Определение стандартов и стиля интерфейса с привлечением пользователей.
- Проектирование и разработка справочной системы и документации.
- Привязка интерфейса к контексту использования.
- Итеративная разработка архитектуры ПО.
- Итеративное конструирование ПО с постепенным введением запланированных функций.
- Контроль удобства использования готового ПО.
-

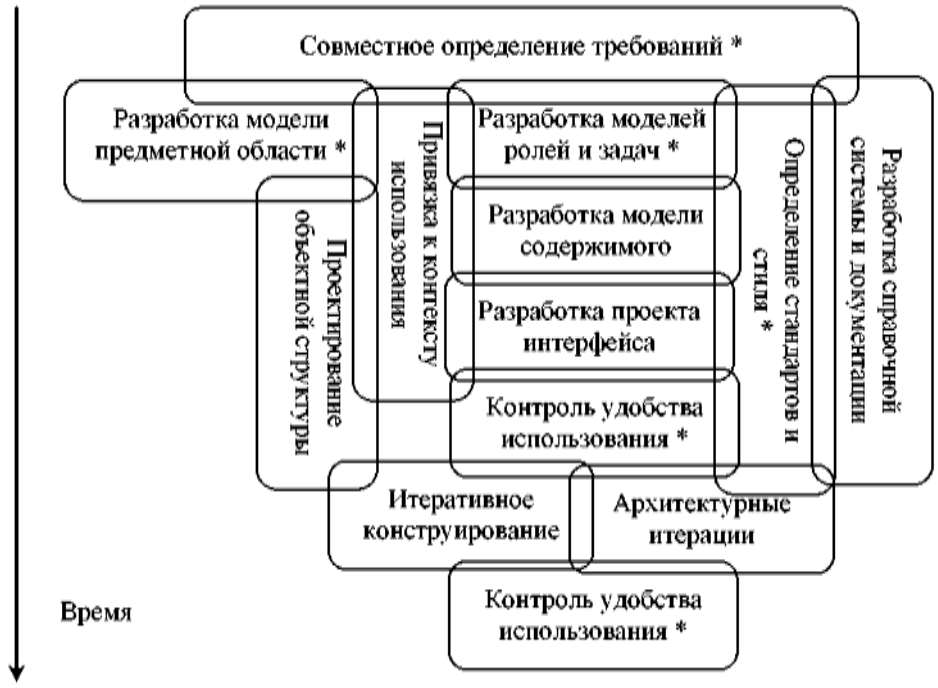

Рисунок 64. Взаимосвязи и распределение деятельностей во времени. Деятельности, в которых вовлечены пользователи, помечены звездочкой.

Эти деятельности не выполняются строго друг за другом в виде отдельных шагов. Для описания их распределения во времени используется диаграмма, изображенная на рис. 64.

Контроль удобства программного обеспечения

Наиболее широко для контроля удобства использования ПО применяются различные виды *инспектирования.*

Эвристическое инспектирование организуется как систематическая оценка различных элементов и аспектов интерфейса с точки зрения определенных эвристик. В качестве таких эвристик можно использовать изложенные выше правила и принципы построения удобных в использовании интерфейсов, или любую другую достаточно полную систему эвристик, приводимых в руководствах по удобству использования ПО.

В рамках одного сеанса инспектирования оценка интерфейса проводится несколькими специалистами, имеющими опыт в деятельности такого рода. Число оценщиков варьируется от 3-х до 5-ти. Их результаты объединяются в общую картину. В процессе инспектирования разработчики должны отвечать на вопросы, касающиеся различных аспектов как предметной области, так и работы проверяемой программы.

Оценка проводится в два этапа. На первом исследуется архитектура интерфейса в целом, на втором — отдельные контексты взаимодействия и элементы интерфейса. В целом оценка занимает 1-3 часа, после чего проводится анализ полученных результатов, во время которого оценщики описывают обнаруженные ими проблемы и предлагают способы их устранения.

При других видах инспектирования могут использоваться различные роли в группе оценщиков (оценщики, ведущий, летописец, пользователи, разработчики, эксперты в предметной области), различные шкалы серьезности обнаруженных дефектов (не более 3-4-х уровней).

В качестве метрик удобства использования применяются, например, следующие показатели (выбраны одни из самых простых и применимые в рамках описанного выше проектирования, ориентированного на удобство использования).

• *Сущностная эффективность* показывает степень приближения производительности пользователей при работе с данным интерфейсом к некоторой идеальной. Определяется она как процентное отношение количества действий, выполняемых пользователем в идеале — в рамках сущностного варианта использования, — к количеству действий пользователя в соответствующем сценарии работы с данным ПО.

$$EE = \frac{Количество\ действийпользователя\ в\ сущностном\ варианте\ использования}{Количество\ действий\ пользователя\ в\ соответствующем\ \ реальном\ сценарии} \cdot 100\%$$

В качестве элементарных действий пользователя учитываются концептуально целостные единицы взаимодействия, такие как перечисленные ниже.

- Ввод данных в одно поле вместе с нажатием перевода строки или табуляции.
- Выбор поля, объекта или группы объектов (двойным или обычным) щелчком мыши или с помощью клавиатуры.
- Переход от мыши к клавиатуре или обратно.
- Выполнение действия (двойным или обычным) щелчком мыши на каком-либо объекте, с помощью меню или клавиатуры.
- Перетаскивание объекта.

Сущностная эффективность интерфейса, предназначенного для решения многих задач, определяется как сумма произведений сущностных эффективностей выполнения отдельных

задач на частоты их выполнения.

$$EE = \sum_i p_i * EE_i$$

• *Согласованность задач* показывает соответствие между частотами выполнения задач и скоростью их выполнения в данном интерфейсе. Вычисляется она следующим образом.

- Задачи располагаются в порядке убывания частоты их возникновения на практике.
- Оценивается количество действий пользователя, необходимое для выполнения каждой задачи.
- Вычисляется индекс согласованности: для каждой пары задач, если порядок в этой паре в соответствии с трудоемкостью их выполнения совпадает с их порядком по частоте использования, к индексу прибавляется 1, иначе из индекса вычитается 1.
- Итоговая согласованность задач оценивается как процентное отношение индекса согласованности к общему количеству различных пар задач, т.е. к $n(n-1)/2$, где n — число различных задач.

Значение этой метрики колеблется от - 100% (полная несогласованность) через 0% (отсутствие корреляции) до 100% (полная согласованность).

Для контроля эффективности работы пользователя с данным интерфейсом часто используется метод количественной оценки, основанный на выделении целей пользователя, операторов, методов и правил их выбора, в качестве названия которого используется аббревиатура ***GOMS*** (Goals, Operators, Methods, and Selection Rules).

Этот метод применим для оценки эффективности работы только достаточно опытных пользователей и не учитывает возникающих по ходу работы ошибок. Кроме того, он опирается на оценки времени реакции системы, которые очень сложно получить для некоторых видов ПО, например, Web-приложений. Основан GOMS на правилах разбиения задач на отдельные действия пользователя и на таблице заранее определенных длительностей выполнения этих действий. В качестве таких действий рассматриваются следующие.

- Нажатие на любую клавишу на клавиатуре, оценивается в 0.28 с.
- Нажатие на кнопку мыши, оценивается в 0.1 с.
- Перемещение указателя мыши, оценивается в 1.1 с.
- Переход от использования клавиатуры к мыши или обратно, оценивается в 0.4 с.
- Выбор очередного действия, оценивается в 1.2 с. Обычно считается, что выбор совершается при каждом переходе фокуса действий пользователя от одного элемента интерфейса к другому.
- Время реакции системы, оценивается в зависимости от имеющихся данных, как минимум в0.1 с. Время реакции системы при выборе пункта меню или элемента раскрывающегося списка обычно не учитывается, но учитывается время открытия окон.

Помимо перечисленных оценочных методов используется и *тестирование удобства использования,* но, по сравнению с ними, оно может применяться только на поздних этапах

разработки и, обнаруживая отдельные проблемы, не дает указаний на возможные исправления или более важные недостатки проекта в целом.

Тестирование проводится обычно в отдельном помещении, в котором пользователь может целиком сосредоточиться на работе с программой. Кроме того, все действия пользователя, ответные реакции системы и реакции пользователя на то, что он видит, протоколируются. Для этого удобно использовать съемку на видеокамеру со спины пользователя, так, чтобы были видны его действия и происходящее на экране. Для фиксации реакций пользователя можно установить зеркало, с помощью которого та же камера может снимать выражение его лица. Это помогает пользователю впоследствии объяснить, чем именно были вызваны его затруднения в данном месте. Кроме того, для протоколирования событий, которые видеокамера может и не зафиксировать, необходимо присутствие наблюдателей-людей, которые, однако, никак не должны влиять на действия пользователя (даже похмыкиванием, вздохами или ерзаньем на стуле, что пользователь может истолковать, часто обосновано, как какие-то намеки на «неправильность» его действий, или наоборот, одобрение).

Пользователь, участвующий в тестировании, должен чувствовать себя раскованно и понимать, что проводимое мероприятие никак не связано с оценкой его профессионализма, знаний или навыков. Это необходимо объяснять, поскольку большинство людей в такой ситуации так или иначе увязывают свои действия с их возможной оценкой окружающими, что вредит адекватности тестирования.