2. МЕТОДИКА РАЗРАБОТКИ ИНТЕРФЕЙСОВ

2.1 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. При таком 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. Основные этапы разработки проблемно-центрированного дизайна:

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

На этапе **анализа задач и пользователей** предстоит определить, кто и зачем собирается использовать разрабатываемую систему. Осведомлѐнность 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. Такие различия среди пользователей, как уровень их квалификации, интерес к изучению новых систем, заинтересованность в успехе разработанной системы, могут обусловливать 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 персонального контакта между командой разработчиков и теми людьми, которые в дальнейшем будут пользоваться системой.

После выработки хорошего понимания пользователей и их задач более традиционный процесс разработки может абстрагироваться от этих фактов и привести к общей спецификации системы и еѐ пользовательского интерфейса. Проблемно-центрированный дизайн предполагает другой 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 идѐт 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. Отобранные задачи должны достаточно полно покрывать всю функциональность системы, и дизайнеру полезно сделать проверочный список всех функций и путѐм сопоставления его с перечнем задач удостоверится, что желаемое покрытие достигнуто.

На этапе **заимствования** необходимо 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. Например, если они используют электронные таблицы, то, может быть, ваш дизайн должен выглядеть как электронная таблица. Существующую парадигму будет легче 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. Например, пусть спецификация вашей системы 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 лучше, если в вашей системе данный интерфейс будет построен таким же образом.

Черновое описание разрабатываемой вами системы должно быть положено на бумагу (обязательно). Это описание не следует оформлять в виде компьютерной программы (пока), даже если вы умеете пользоваться какимилибо системами автоматизации разработки. Такие системы вынуждают вас прикрепляться к конкретным решениям, которые ещѐ 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. То есть, если предлагается введение новой возможности, то необходимо определить, 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.

Никакая авиационная компания не будет разрабатывать и строить реактивный самолѐт 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. Существует несколько структурных подходов, которые можно использовать, чтобы исследовать сильные и слабые стороны интерфейса до его программного воплощения. Данный этап называется этапом **обдумывания дизайна**. Один из методов состоит в подсчѐте количества нажатий клавиш, движений мыши и мыслительных операций (решений), необходимых для выполнения задач, предписанных разрабатываемой системе. Это позволяет оценить трудоѐмкость выполнения задач по времени и выявить задачи, требующие слишком много шагов. Такой метод называется GOMS анализом. Другой метод основан на приѐме, 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.

После этапа обдумывания дизайна по его описанию на бумаге, приходит очередь **создания макета или прототипа интерфейса**, представляющего собой дальнейшую детализацию предстоящей работы, которую можно показать пользователям. Для этого могут использоваться специальные инструменты для создания прототипов, которые позволяют отделить графическую часть интерфейса от логики функционирования системы. На данном этапе не нужно реализовывать систему целиком. Усилия должны быть сосредоточены на частях еѐ интерфейса, нужных для решения репрезентативных задач.

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 приѐм "думать вслух". Вы просите пользователя не только делать необходимые действия для решения поставленной задачи, но и

говорить, что он делает и о чѐм размышляет. Эти комментарии необходимо записывать, так как они дают неоценимые данные для улучшения дизайна системы. Информация, собранная при тестировании системы с пользователями, даѐт 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 еѐ предполагаемому пользователю.

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 серьѐзностью возникших проблем, затем доработать интерфейс и протестировать его снова. Серьѐзные проблемы могут даже потребовать пересмотра понимания задач и пользователей, т.е. откатить вас на первый этап проблемно-центрированного подхода. Необходимо на каждой **итерации** помнить, что 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 приѐмов. Другие же не выявятся без тестирования с пользователями. Когда следует прекратить итерации? Если были установлены специфические требования практичности, 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.

Ключевой руководящий принцип в 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 пишите собственную подпрограмму для реализации специализированного меню, то не закладывайте жѐстко 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. Всѐ 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.

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. **Отслеживание эксплуатации** — это ключевой момент для всех серьѐзных организаций, заинтересованных в своѐм постоянном присутствии на рынке. Один из методов введения разработчиков в контакт с пользователями – организация поочерѐдных дежурств на горячей линии связи с потребителями. Другая важная вещь для больших систем – организация собраний групп пользователей и конференций. Такая работа позволяет лучше понять реакцию пользователей на продукт, который они продают. Эта информация в дальнейшем позволяет улучшить описание задач для новой версии продукта и углубить понимание разработчиком предметной области.

На существующем сегодня рынке компьютеров и программ вряд ли существуют продукты, которые поддерживают свою жизнеспособность без регулярных усовершенствований. Независимо от того, насколько удачно система была спроектирована первоначально, с большой вероятностью она будет терять адекватность с течением лет. Меняются задачи, меняются пользователи, приходит время **изменения дизайна**. Приѐмы 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.

Практичность (англоязычный термин – usability) – это одна из важнейших характеристик систем, изучению которой посвящено множество исследований. Управленческая деятельность в серьѐзной 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 определѐнных 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.

2.2 CWT-анализ интерфейса.

CWT анализ – это формализованный способ представления мыслей и действий людей, когда они пользуются интерфейсом в первый раз. Аббревиатура CWT означает Cognitive Walkthrough (познавательный сквозной контроль). Всѐ начинается с того, что у вас есть прототип или детальное описание интерфейса, и вы знаете, кто будет пользователем системы. Вы выбираете одну из задач, решение которых интерфейс должен поддерживать, затем формируете полный письменный список действий, необходимых для выполнения задачи при помощи интерфейса. Далее вы пытаетесь рассказать «правдоподобную историю» о каждом действии, которое должен выполнить пользователь. Для этого необходимо давать мотивацию каждого действия пользователя исходя из его предполагаемых знаний и подсказок и реакций интерфейса. Для каждого действия могут обнаруживаться проблемы, мешающие правильному его выполнению. Эти проблемы записываются, но затем вы предполагаете, что они исправлены, и переходите к следующему действию. В результате у вас остаѐтся список проблем, что является руководством к действию по исправлению (улучшению) интерфейса.

CWT-анализ позволяет обнаружить несколько типов проблем с интерфейсом.

1. Поставить под сомнение ваши первоначальные и не вполне обоснованные предположения о том, как мыслит пользователь.
2. Выявлять элементы управления, которые очевидны для разработчика, но могут быть непонятны пользователю.
3. Выявлять затруднения с надписями и подсказками.
4. Обнаруживать неадекватную обратную связь, что может заставить пользователя сомневаться в результате и повторять всѐ с начала, хотя всѐ было сделано правильно.
5. Показывать недостатки в текущем описании интерфейса.

CWT-анализ фокусируется в основном на проблемах, которые пользователи испытывают при первом взаимодействии, не проходя предварительных тренировок. Такая постановка вопроса чрезвычайно важна для некоторых критических систем, таких как банкоматы или терминалы оплаты. Но та же самая ситуация возникает и в сложных программных системах, когда пользователь выполняет какую-либо задачу впервые. Пользователи часто изучают сложные программы постепенно, углубляясь в детали интерфейса по мере возникновения в том необходимости. Поэтому проблемы "первого знакомства" могут возникнуть даже при взаимодействии с давно используемой программой. Если система построена с уважением принципов CWT-анализа, то она позволяет пользователю плавно подниматься с уровня новичка до уровня эксперта.

Многие упускают необходимость сформировать полный и точный список действий для выполнения задачи. То есть они сами точно не знают, как выполнить задачу, и блуждают по интерфейсу, пытаясь отыскать правильную последовательность действий, а потом они оценивают сложность этого блуждания. Иногда, действительно, бывает полезно оценить сложность такого блуждания, но это не метод CWT. В CWT вы должны начинать, имея в руках полный список отдельных элементарных действий, необходимых для выполнения задания. Если в ходе анализа выясняется, что пользователь испытывает затруднения в определении и выполнении одного из действий, то вас интересует не то, как пользователь выйдет из положения, а сам факт того, что проблема возникла и интерфейс нуждается в доработке.

Основные рекомендации по проведению CWT-анализа сводятся к следующему. Вы определили задачу, класс пользователей, интерфейс и корректную последовательность действий. Далее рекомендуется собрать вместе группу разработчиков и других заинтересованных лиц (в учебных целях вы можете действовать в одиночку). И после этого начинается процесс анализа. Вы пытаетесь рассказать историю о том, почему пользователь выбрал бы каждое действие в списке корректных действий. Вы критикуете историю, чтобы сделать еѐ правдоподобной. Рекомендуется держать в уме четыре вопроса:

- Будут ли пользователи пытаться произвести тот или иной эффект, который даѐт действие?
- Видят ли пользователи элемент управления (кнопку, меню, переключатель и т.д.) для осуществления действия?
- Если пользователи нашли элемент управления, поймут ли они, что он производит тот эффект, который им нужен?

• После того как действие сделано, будет ли понятен пользователям тот отклик, который они получают, чтобы перейти к следующему действию с уверенностью?

По результатам CWT-анализа необходимо исправить интерфейс. Большинство исправлений будут очевидны. Сложный случай возникает тогда, когда у пользователя вообще нет никаких причин думать, что действие должно быть сделано. Самое верное решение этой проблемы – исключить это действие; пусть система сама выполнит его. Если так не получается, то необходимо перестроить задачу так, чтобы пользователи получали бы логическое побуждение к выполнению "проблемного" действия.

CWT-анализ позволяет выявить много проблем с интерфейсом, но не даèт ответа на вопрос, насколько сложен интерфейс, т.е. сколько времени тратит пользователь на выполнение задачи.

2.3 Анализ GOMS.

GOMS анализ оценивает время работы с интерфейсом обученного пользователя. Даже если интерфейс успешно прошел CWT-анализ, это не означает, что он оптимален с точки зрения трудоèмкости. Если есть несколько альтернативных вариантов построения интерфейса, то анализ GOMS позволяет выбрать тот из них, который требует меньше времени для решения задачи пользователя. В отличие от CWT, GOMS предполагает, что пользователь уже знает интерфейс, т.е. видит его не первый раз.

Аббревиатура GOMS означает Goals, Operations, Methods, Selections (цели, операции, методы, правила выбора). Модель GOMS состоит из описания методов, необходимых для достижения заданных целей. Методы представляются последовательностями шагов, состоящих из операций, которые выполняет пользователь. Метод может быть назван подцелью, т.о. методы образуют иерархическую структуру. Если существует более одного метода для достижения цели, то включаются правила выбора, с помощью которых в зависимости от контекста выбирается один метод.

В терминологии данного вида анализа цель – это то, что мы раньше называли (репрезентативной) задачей, метод – это список действий пользователя, операции – элементарные действия пользователя.

Операции в GOMS – это элементарные действия, которые нельзя разложить на более мелкие. Причѐм учитываются как внешние действия, т.е. те, что приводят к видимым физическим эффектам, так и внутренние, связанные с мышлением пользователя. Например, действие (шаг метода) "нажать кнопку <button>" представляется в виде последовательности следующих операций:

- визуально определить местонахождение кнопки (мыслительная операция);
- навести на кнопку указатель мыши (внешняя операция);
- щелкнуть кнопкой мыши (внешняя операция).

Рассмотрим анализ интерфейсов для типичной конфигурации персонального компьютера, где в качестве устройств ввода-вывода выступают монитор, клавиатура и мышь. Практически все интерфейсные взаимодействия в этом случае можно описать следующими операциями:

K – нажатие клавиши;

B – клик кнопкой мыши;

P – наведение указателя мыши;

R – ожидание ответной реакции компьютера;

H – перенос руки с клавиатуры на мышь или наоборот;

D – проведение с помощью мыши прямой линии (например, выделение или прокрутка текста);

M – мыслительная подготовка (к осуществлению одной из перечисленных операций).

Разные пользователи выполняют указанные операции за разное время. Однако, напомним, что GOMS исследует работу опытного пользователя. Многочисленные исследования выявили средние значения времени операций, выполняемых опытными пользователями. Приведѐм их значения:

K 0.2 с
B 0.2 с
P 1.1 с
H 0.4 с
M 1.35 с

Время выполнения задачи, посчитанное с использованием этих значений, является хорошей средней оценкой сложности интерфейса и действительно работает на практике.

Время ожидания *R* зависит от характера действия, выполняемого компьютером. Речь идѐт только о тех задержках, которые связаны с работой интерфейса (например, ожидание открытия нового интерфейсного объекта). При анализе GOMS, как правило, не учитывается время, расходуемое компьютером на выполнение целевой вычислительной функции (например, преобразование одного формата в другой). Такие вычислительные операции относятся уже к функциональному программному обеспечению; скорость их выполнения определяется быстродействием аппаратуры, объѐмом обрабатываемых данных и

эффективностью алгоритмов, но не связана со сложностью интерфейса. С другой стороны, очень быстрые реакции компьютера, кажущиеся для человека практически мгновенными (например, эхо-печать символа после нажатия клавиши), также не учитываются. Следует учитывать только ожидания, которые сбивают ритм выполнения других операций. Время таких ожиданий можно оценить при работе с реальной программой следующим образом: Если реакция компьютера достаточно быстрая, но, всѐ же, ощутима, то "стандартное" время *R* рекомендуется установить 0.25 с.

Время операции *D* также может быть разным в зависимости от величины перемещения и других сопутствующих деталей. Его рекомендуется приблизительно оценить при работе с реальным приложением. В качестве "стандартного" значения можно взять 2 с.

Общий алгоритм действий при проведении GOMS анализа:

- Цель разбивается на подцели,
- для каждой подцели расписываются методы,
- методы могут раскрываться далее через внутренние методы Формируется конечная последовательность операций добавляются мыслительные подготовки *M*.
- Считается время в секундах

Вся последовательность операций разбивается на семантические группы. Например, набор на клавиатуре слова "слон" выполняется четырьмя операциями нажатия на клавиши *KKKK* и рассматривается как отдельная семантическая единица. Перед ней нужно поставить операцию *M*. Действительно, перед тем как напечатать слово, человек должен сформировать его внутренний образ. Это и отражается добавлением операции *M*. Но после того, как образ сформирован, слово печатается без дальнейших раздумий между отдельными буквами. Пример другой семантической единицы – открытие меню. Оно состоит из двух операций *PB*. Но прежде, чем они могут быть выполнены, нужно решить, какой пункт меню вам нужен и найти, где он находится. Поэтому перед *PB* (но не между *P* и *B*) нужно поставить *M*. Если затем с помощью ещѐ одной пары операций *PB* в меню выбирается элемент, то *M* ставить не нужно, т.к. идея о том, что нужно выбрать из меню, уже сформировалась у человека ранее.

Наконец, когда вся последовательность операций выписана, мы получаем общее время выполнения (оценка среднего времени) задачи простым суммированием времѐн отдельных операций. Но это не единственный результат. Проведѐнный анализ часто позволяет понять, как можно улучшить интерфейс для сокращения времени решения задачи.

Рассмотренных операций может оказаться недостаточно для некоторых интерфейсов. Например, если вы используете сенсорные панели или графические планшеты, то понадобятся специальные, связанные с ними операции. Их временные характеристики можно определить экспериментально или найти в специальной литературе.

2.4 Золотые правила построения интерфейсов

За годы изучения проблем человеко-машинного взаимодействия, специалисты выявили несколько наиболее существенных правил построения интерфейсов, и назвали их "золотыми правилами". Эти правила могут также использоваться для экспертной оценки существующих интерфейсов.

2.4.1 Правила Нильсена Молиха (Nielsen, Molich).

Простой и естественный диалог. Простота означает, что не должно присутствовать не относящейся к теме или редко используемой информации. Старайтесь добиваться максимального следования задачам пользователя, минимизируя сложность поиска соответствия между семантиками задачи и интерфейсом. Важно представить точно ту информацию, в которой нуждается пользователь, следуя принципу "лучше меньше да лучше". Вся редко используемая информация должна быть спрятана. Естественность означает, что информация, которая выводится на экран, должна появляться в естественном порядке, соответствующем ожиданиям пользователя. Вся взаимосвязанная информация должна группироваться в одном месте.

Говорите на языке пользователя. Используйте слова и понятия из мира пользователя. Не используйте специфических инженерных терминов. Например, для большинства людей представление цвета в виде RGB-компонент и их шестнадцатеричная кодировка являются совершенно непонятными вещами. Лучшим и довольно простым решением будет показать вместо кода (или наряду с ним) образец цвета.

Минимизируйте загрузку памяти пользователя. Не заставляйте пользователя помнить вещи от одного действия к следующему. Оставляйте информацию на экране до тех пор, пока она не перестанет быть нужной. Например, хорошим стилем считается делать только один ряд закладок.

Будьте последовательны. У пользователей должна быть возможность изучить действия в одной части системы и применить их снова, чтобы получить похожие результаты в других местах.

Обеспечьте обратную связь. Дайте пользователю возможность видеть, какой эффект оказывают его действия на систему.

Обеспечьте хорошо обозначенные выходы. Если пользователь попадает в часть системы, которая его не интересует, у него всегда должна быть возможность быстро выйти оттуда, ничего не повредив.

Обеспечьте быстрые клавиши и ярлыки. Элементы быстрого доступа могут помочь опытным пользователям избегать длинных диалогов и информационных сообщений, которые им не нужны.

Хорошие сообщения об ошибках. Хорошее сообщение об ошибке помогает пользователю понять, в чѐм проблема и как это исправить.

Предотвращайте ошибки. Всегда, когда вы пишете сообщение об ошибке, вы должны спросить себя, можно ли избежать этой ошибки? Хороший пример построения интерфейса, предотвращающего ошибки – стандартная программа «Калькулятор», в которой при переключении в двоичный режим интерфейс делает недоступными числовые кнопки, кроме 0 и 1, а также делает недоступными некоторые функции, имеющие смысл только для чисел с плавающей точкой. Этот простой приѐм позволяет минимизировать возможные ошибки пользователя.

Снабдите программу системой помощи. Справка должна быть максимально подробной и рассчитана на абсолютно неопытного пользователя.

2.4.2. Принципы организации графического интерфейса

Принцип кластеризации. Организуйте экран в виде визуально разделённых блоков с похожими элементами управления, предпочтительно с названием для каждого блока. Элементы управления включают меню, диалоговые боксы, экранные кнопки и любые другие графические элементы, позволяющие пользователю взаимодействовать с компьютером. Подобные команды должны быть в одном меню: это позволяет им быть визуально близко и идти под одним заголовком. Команды, относящиеся к некоторой конкретной области функциональности, могут также быть показаны в диалоговых боксах, в визуально определимых блоках. Тот же самый принцип распространяется на специальные управляющие экраны с многочисленными кнопками и значками, такие как сенсорные панели. Кнопки для конкретной функции должны группироваться вместе, а затем выделяться цветом, обрамлением или окружающим пробелом.

Существует две важные причины для кластеризации. Во-первых, она помогает пользователю находить нужную команду. Если вы ищете возможность изменить формат абзаца, то вам легче найти соответствующий диалоговый бокс в меню "Формат", а не в случайно распределённом по экрану наборе из сотни кнопок. Во-вторых, группирование команд позволяет пользователю приобрести концептуальную организацию знаний о программе. Полезно, например, знать,

что начертание и размер являются атрибутами шрифта, в то время как интервал и отступ – атрибутами абзаца.

Принцип "видимость отражает полезность". Делайте часто используемые элементы управления заметными, видимыми и легко доступными; и наоборот, прячьте или сжимайте редко используемые элементы. Пример реализации этого принципа в современных системах – панели инструментов, т.е. наборы иконок для часто используемых функций. Обоснование этого принципа заключается в том, что человек может быстро находить что-то среди небольшого числа объектов. Для редко используемых функций можно позволить поиск в длинных списках.

Принцип интеллектуальной последовательности. Используйте похожие экраны для похожих функций. Это похоже на заимствование, но в этом случае вы заимствуете что-то из одной части дизайна и применяете это к другой части. Обоснование этого принципа очевидно: раз пользователи выучили расположение элементов управления на экране (например, где находится кнопка "Помощь"), они могут легко приложить это знание к другим экранам внутри той же системы. Этот подход позволяет вам сосредоточить усилия на разработке лишь нескольких привлекательных работоспособных экранов, а затем их слегка модифицировать для использования в других частях приложения. Но экраны не должны выглядеть одинаково, если в действительности они должны отражать совершенно разные вещи. Предупреждение о критической ошибке в системе реального времени должно иметь вид, значительно отличающийся от экрана помощи или информационного сообщения.

Принцип "цвет как приложение". Не полагайтесь на цвет как носитель информации; используйте его умеренно, чтобы лишь акцентировать информацию, передаваемую другими средствами. Хорошее правило для большинства интерфейсов – разрабатывать их в чѐрно-белом варианте, убедиться, что они работоспособны, а затем добавить минимальный цвет для их окончательного облика. Цвет, безусловно, полезен, когда необходимо выделить предупреждение или информационное сообщение, но обязательно дайте дополнительные ключи для пользователей, не способных воспринимать изменения цвета.

Принцип уменьшения беспорядка. Не помещайте на экран слишком много всего. Этот неформально определяемый принцип – хорошая проверочная точка, чтобы подтвердить, что ваш дизайн отражает перечисленные выше принципы. Если видимы только наиболее часто используемые элементы, все они сгруппированы в небольшое число визуальных кластеров, использован минимум цвета, то экран должен получиться графически привлекательным. Не пытайтесь наделить каждое меню собственным шрифтом или работать с большим набором

размеров. Как правило, пользователи заметят не столько различия, сколько беспорядок.

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1) Назовите основные этапы построения интерфейса.
2) Как происходит тестирование дизайна приложений?
3) В чем заключается принцип CWT-анализа?
4) Как и для чего проводят GOMS-анализ?
5) Что включают в себя золотые правила построения интерфейсов?