Принципы разработки пользовательского интерфейса.

Одним из важных показателей качества ПО является удобство его использования. Для построения действительно удобного ПО необходим учет психологии пользователей, хотя роль психологических факторов меньше, чем значимость того, помогает ли ПО решать действительно значимые задачи, или нет.

В понятие удобства использования входят такие характеристики ПО, как понятность пользовательского интерфейса, легкость обучения работе с ПО, трудоемкость решения определенных задач с его помощью, производительность работы пользователя с ПО, частота появления жалоб на неубоства.

В последнее время большое значение в плане обеспечения удобства использования придают стандартизации внешнего вида пользовательского интерфейса и основных элементов, с помощью которых пользователь контактирует с ПО. Такая стандартизация действительно повышает удобство работы в том аспекте, что пользователям становится легче ориентироваться в незнакомой программе. Однако значительно повысить удобство использования, особенно для опытных пользователей, только с помощью применения стандартных элементов интерфейса чаще всего не удается.

При рассмотрении задач построения удобного в использовании ПО иногда привлекают множество информации из психологии когнитивных процессов, изучающей восприятие информации людьми. Похоже, что наиболее полезные для разработки пользовательского интерфейса результаты психологических исследований можно сформулировать достаточно кратко.

- Человеку свойственно ошибаться.
  Поэтому удобная система должна быть достаточно термима к ошибкам пользователей и нив коем случае не рушится из-за ошибок ввода.
- Человек осознает информацию и производит действия достаточно медленно по меркам компьютеров.
  Около 0.1 секунды требуется для распознавание визуального объекта, около 0.25 секунды — на перевод взгляда и переключение внимания с одного объекта на другой, около 1.25 секунды — на принятие простейшего решения (выбор из двух альтернатив).
  Эти цифры должны учитываться при проектировании интерфейсов, рассчитанных на взаимодействие с человеком.
- Глаз быстрее руки — человек быстрее узнает что-то, чем производит действия.
  На нажатие клавиши на клавиатуре или кнопки мыши уходит от 0.1 до 1.25 секунды, на наведение указателя мыши на объект — 1.5 секунды, на переключение внимания с мыши на клавиатуру уходит около 0.36 секунды.
  Кроме того, скорость переноса указателя мыши зависит от дальности переноса D и размеров W объекта, на который нужно попасть по закону Фитса (Fitts) — $T = a + b*\log(D/W)$ (иногда $a + b*(D/W)^{1/2}$)
  В любом случае — чем больше объект, тем быстрее можно установить на него указатель. Кроме того, люди чаще всего промахиваются при первой попытке переместить указатель мыши.
- Правило 7±2.
  Человек способен хранить в кратковременной памяти одновременно не более 5-9 отдельных сущностей. Существуют исключения из этого правила, но практически

всегда разбираться в картинке, где изображено больше 10 объектов труднее, чем в той, где их меньше — ее тяжело охватить в целом.

- Человек гораздо быстрее узнает что-то, чем вспоминает, как оно называется. Значительно проще выбрать что-то из списка, чем набрать его идентификатор или имя.
- На изучение нового всегда требуется время. В начале работы с чем-то новым человек гораздо чаще совершает ошибки, даже если пытается решать уже известные ему задачи с помощью нового инструмента.
- Внимание человека прежде всего акцентируется на движущихся объектах, затем — на выделяющихся цветом, и только потом на остальных формах выделения. Лучшим способом отвлечь человека от работы является появление цветных анимированных ненужных картинок.

Основные правила построеняи удобного ПО могут быть сформулированы так.

- Правило доступности.
  Система должна быть настолько понятной, чтобы пользователь, никогда раньше не видевший ее, но хорошо разбирающийся в предметной области, мог без помощи инструкторов начать ее использовать.
  Это правило, в основном, служит лишь указанием идеала, к которому надо стремиться, поскольку на практике достичь такой понятности почти никогда не удается.
- 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.
- Правило непрерывного движения вперед.
  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 сложными воможностями, которыми 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.
- Правило поддержки.
  Система должна способствовать более простому и быстрому решению задач пользователя.
  Это означает, прежде всего, что система должна *действительно решать* задачи пользователя.
  Во-вторых, она должна решать их *лучше, проще и быстрее*, чем имевшиеся до ее появления методы.
- Правило соблюдения контекстов.
  Система должна быть согласована со средой, в которой 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, и пр.

Представленные правила определяют общие требования к разрабатываемому интерфейсу. Как можно действовать, чтобы соблюсти эти требования, в общих чертах можно усвоить на основе следующих принципов.

- Принцип структуризации.
  Пользовательский интерфейс должен быть целесообразно структурирован. Родственные его части должны быть связаны, а независимые — разделены, похожие элементы должны выглядеть похоже, а непохожие — различаться.
- Принцип простоты.
  Наиболее распространенные операции должны выполняться максимально просто. При этом должны быть ясные ссылки на более сложные процедуры.
- Принцип видимости.
  Все функции и данные, необходимые для выполнения определенной задачи, должны быть видны, когда пользователь пытается ее выполнить.
- Принцип обратной связи.
  Пользователь должен получать сообщения о действиях системы и о важных событиях внутри нее. Сообщения должны быть 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 и данных. По возможности, следует избегать модального взаимодействия.
- Принцип повторного использования.
  Следует стараться использовать многократно внутренние и внешние компоненты, способствуя тем самым унифицированности интерфейса.

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

Основной идей этого подхода является использование специальных моделей, ориентированных на описание и анализ использования ПО. Главная цель таких моделей — получить представление о том, какие задачи пользователи решают с помощью данного ПО и как должна быть организована информация для обеспечения решения этих задач.

Список моделей, с помощью которых предалагется проектировать пользовательский интерфейс, представлен ниже.

- Модель ролей.
  Эта модель дает список ролей пользователей системы, каждая из которых представляет собой абстрактную группу задач и нужд некоторого множества пользователей. Ролевая модель может определять связи между ними — по уточнению, по включению, по сходству — и набор из одной-двух-трех центральных ролей, на которых, 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 быть следующими.
    - Обязанности — требования к знаниям (о предметной области, о самой системе и прочим), которым пользователь в данной роли, скорее всего, удовлетворяет.

- Умения — уровень мастерства в работе с системой.
- Взаимодействия — типичные варианты взаимодействия с системой, включая их частоту, регулярность, непрерывность, концентрацию, интенсивность, сложность, предсказуемость, управление взаимодействием.
- 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.
- Другие — возможные убытки от ошибок, риски и пр.

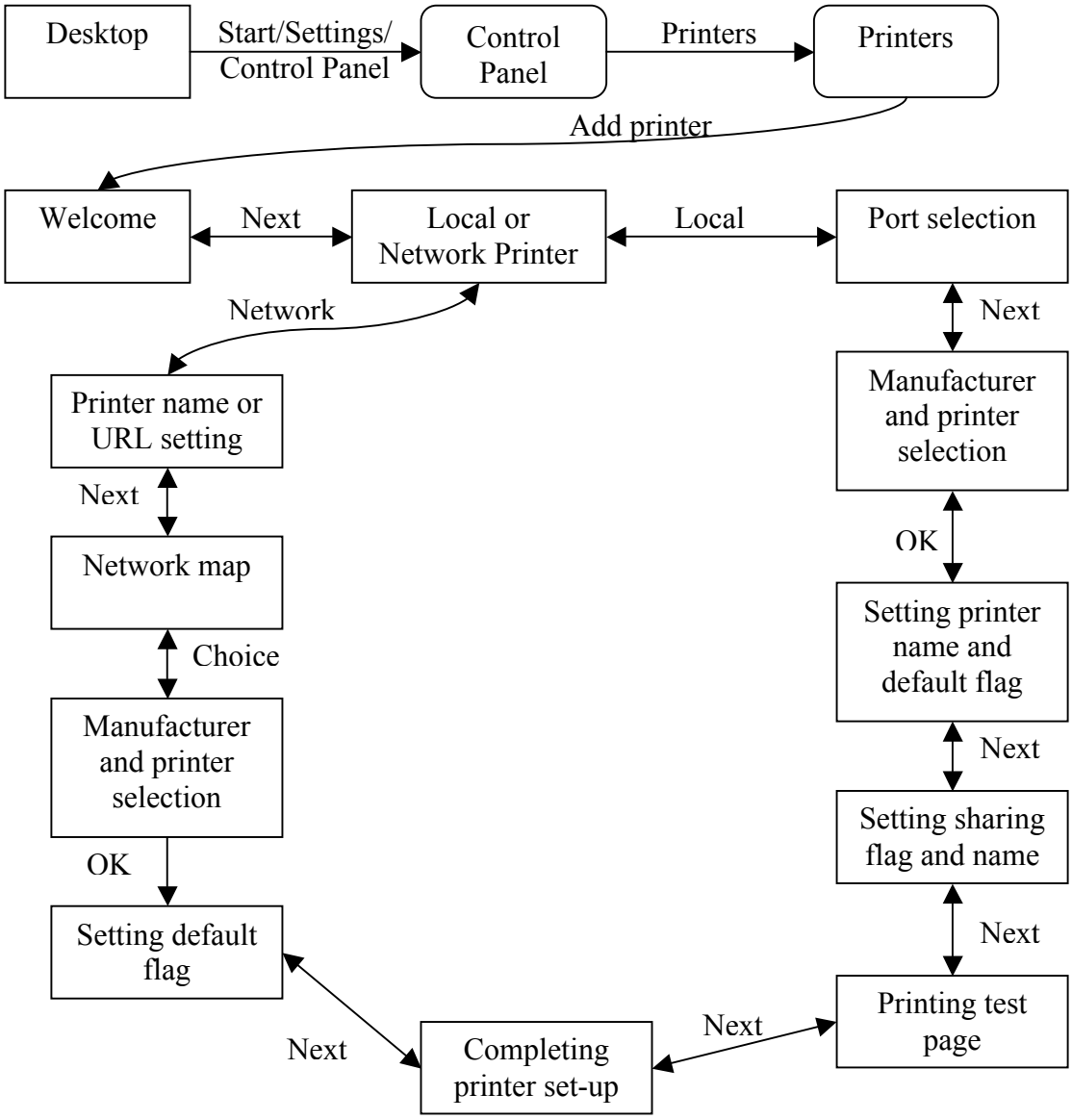
- Модель задач.
  Модель задач, которую можно использовать для проектированяи пользовательского интерфейса, строится на основе сценариев или вариантов использования (use cases), в которых остаются только цели и задачи пользователя в рамках данного варианта, и нет неявных предположений о наличии определенных интефейсов между пользователями и системой. Удобно описывать такие сценарии в виде двух наборов — устремлений пользователя (не действий, а задач которые он хочет решить) и обязательств системы в ответ на эти устремления.
  Так, например, сценарий работы пользователя с банкоматом выглядит в обычной записи и в нацеленной на выделение задач так.

Действия	**Реакции**	**Задачи**	**Обязательства**
Встаить карту		Регистрация в системе	
	Считать данные		
	Запросить PIN		
Ввести PIN			
	Проверить PIN		Проверка личности
	Вывести меню		Предложение выбора
Нажать клавишу операции выдачи денег		Выбор операции выдачи	
	Запросить сумму		
Ввести сумму			
	Вывести сумму		
	Запросить подтверждение		
Нажать клавишу подтверждения			
	Вернуть карту		
	Выдать деньги		Выдача денег
	Напечатать чек		
	Выдать чек		Выдача чека

  В результате модель задач представляет собой набор переработанных вариантов использования со связями между ними по расширению и по использованию.

Всякая пользовательская роль должна быть поддержана 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 важно определить что войдет в состав интерфейса и не решать вопрос о том, как оно будет выглядеть.
  На начальном этапе один контекст взаимодействия ставится в соответствие одному (не вспомогательному!) варианту использования или группе очень походих вариантов, для выполнения которых понадобится один набор инструментов.
  Средства для поддержки вспомогательных расширяющих вариантов использования обычно удобно помещать в контексты расширяемых ими основных вариантов.
  Сначала определяется, какая информация должна находится в заданном контексте для успешного решения задач соответствующего варианта использования, затем определяется список необходимых операций с этой информацией.
  После определения набора контекстов и их информационного и функционального содержимого рисуется *карта навигации* между контекстами.
  После разработки модели содержимого всякий основной вариант использования должен быть поддержан при помощи одного или нескольких контекстов.
  Перечисленные три модели — ролей, задач и содержимого — являются основными. Оставшиеся две модели используются не всегда.



Пример карты навигации для установки нового принтера в Windows.

Литература

[1] Л. Константайн, Л. Локвуд. Разработка программного обеспечения. Питер, 2004.

[2] Р. Дж. Торрес. Практическое руководство по проектированию и разработке пользовательского интерфейса. Вильямс, 2002.

[3] Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования. Питер-ДМК, 2001.

[4] Э. Дж. Брауде. Технология разработки программного обеспечения. Питер, 2004.