Содержание:

ВВЕДЕНИЕ

Слово «интерфейс» заимствовано из английского языка, где буквально означает «между лицами», т.е. используется в значениях: «взаимодействие, разделение, внешний вид». В современной IT-сфере интерфейсом называют унифицированные системы связи, обеспечивающие обмен информацией между различными объектами.

Это понятие наиболее часто используется в компьютерной технике, но нередко употребляется и в других технических областях, а также в инженерной психологии, где означает различные способы коммуникации между человеком и машиной.

Цель данной работы – рассмотрение процесса построения интерфейсных программ.

Для достижения поставленной цели в ходе работы, будут решены следующие задачи:

1.Рассмотрено, что представляет из себя «Интерфейс».

2. Рассмотрены принципы построения интерфейсных программ.

1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ПОСТРОЕНИЯ ИНТЕРФЕЙСНЫХ ПРОГРАММ

1.1 Сущность интерфейсов

Интерфейс — программная/синтаксическая структура, определяющая отношение между объектами, которые разделяют определенное поведенческое множество и не связаны никак иначе. При проектировании классов, разработка интерфейса тождественна разработке спецификации (множества методов, который каждый класс, использующий интерфейс должен реализовывать).

Интерфейсы, наряду с абстрактными классами и протоколами, устанавливают взаимные обязательства между элементами программной системы, что является фундаментом концепции программирования по контракту. Интерфейс определяет границу взаимодействия между классами или компонентами, специфицируя определенную абстракцию, которую осуществляет реализующая сторона.

Интерфейс в ООП является строго формализованным элементом объектно-ориентированного языка и широко используется в исходном коде программ.

Описание ООП-интерфейса, если отвлечься от деталей синтаксиса конкретных языков, состоит из двух частей: имени и методов интерфейса.

- *Имя интерфейса* строится по тем же правилам, что и другие идентификаторы используемого языка программирования. Разные языки и среды разработки имеют различные соглашения по оформлению кода, в соответствии с которыми имена интерфейсов могут формироваться по некоторым правилам, которые помогают отличать имя интерфейса от имён других элементов программы. Например, в технологии COM и во всех поддерживающих её языках действует соглашение, следуя которому, имя интерфейса строится по шаблону «I<Имя>», то есть состоит из написанного с заглавной буквы осмысленного имени, которому предшествует прописная латинская буква I (IUnknown, IDispatch, IStringList и т. п.).
- Методы интерфейса. В описании интерфейса определяются имена и сигнатуры входящих в него методов, то есть процедур или функций класса.

Использование интерфейсов возможно двумя способами:

- Класс может реализовывать интерфейс. Реализация интерфейса заключается в том, что в описании класса данный интерфейс указывается как реализуемый, а в коде класса обязательно определяются все методы, которые описаны в интерфейсе, в полном соответствии с сигнатурами из описания этого интерфейса. То есть, если класс реализует интерфейс, для любого экземпляра этого класса существуют и могут быть вызваны все описанные в интерфейсе методы. Один класс может реализовать несколько интерфейсов одновременно.
- Возможно объявление переменных и параметров методов как имеющих тип «интерфейс». В такую переменную или параметр может быть записан экземпляр любого класса, реализующего интерфейс. Если интерфейс объявлен как тип возвращаемого значения функции, это означает, что функция

возвращает объект класса, реализующего данный интерфейс.

Как правило, в объектно-ориентированных языках программирования интерфейсы, как и классы, могут наследоваться друг от друга. В этом случае интерфейс-потомок включает все методы интерфейса-предка и, возможно, добавляет к ним свои собственные.

Таким образом, с одной стороны, интерфейс — это «договор», который обязуется выполнить класс, реализующий его, с другой стороны, интерфейс — это тип данных, потому что его описание достаточно чётко определяет свойства объектов, чтобы наравне с классом типизировать переменные. Следует, однако, подчеркнуть, что интерфейс не является полноценным типом данных, так как он задаёт только внешнее поведение объектов. Внутреннюю структуру и реализацию заданного интерфейсом поведения обеспечивает класс, реализующий интерфейс; именно поэтому «экземпляров интерфейса» в чистом виде не бывает, и любая переменная типа «интерфейс» содержит экземпляры конкретных классов.

Использование интерфейсов — один из вариантов обеспечения полиморфизма в объектных языках и средах. Все классы, реализующие один и тот же интерфейс, с точки зрения определяемого ими поведения, ведут себя внешне одинаково. Это позволяет писать обобщённые алгоритмы обработки данных, использующие в качестве типов параметры интерфейсов, и применять их к объектам различных типов, всякий раз получая требуемый результат.

Например, интерфейс «Cloneable» может описать абстракцию клонирования (создания точных копий) объектов, специфицировав метод «Clone», который должен выполнять копирование содержимого объекта в другой объект того же типа. Тогда любой класс, объекты которого может понадобиться копировать, должен реализовать интерфейс Cloneable и предоставить метод Clone, а в любом месте программы, где требуется клонирование объектов, для этой цели у объекта вызывается метод Clone. Причём, использующему этот метод коду достаточно иметь только описание интерфейса, он может ничего не знать о фактическом классе, объекты которого копируются.

Совокупность управляющих элементов программы, с помощью которых пользователь выполняет различные действия, называется интерфейсом программы. Говоря простыми словами, интерфейс программы – это те кнопки и окошки, которые вы используете для того, чтобы программа совершала нужные

вам действия.

При создании программного продукта особая роль отводится графическому интерфейсу самого продукта. Графический интерфейс пользователя это тип пользовательского интерфейса, который позволяет пользователю взаимодействовать с электронными устройствами посредством графических изображений вместо ввода текстовых команд.

Например, пользователю привычнее и понятнее двигать ползунок, оформленный в виде ручки и представляющий собой графическое изображение, нежели постоянно вводить числа от единицы до ста. Графические интерфейсы широко используются как в компьютерах, так и в портативных устройствах, таких как MP3-плееры, портативные медиа-плееры или игровые устройства, бытовой технике и в офисном оборудовании.

Например[1]:

Рисунок 1 – Примеры интерфейс программ

Когда проектируешь новый интерфейс, нужно учитывать не только интересы заказчика, у которого в основном одна цель — обеспечить финансовый поток, но и мотивацию пользователей. Когда ты понимаешь, что твой пользователь — не абстрактный бухгалтер, а женщина 60-ти лет, которая хочет пораньше уйти с работы и не учить ничего нового, это сильно влияет на интерфейс. Ты учитываешь, что она не захочет делать, и как её можно заставить — кнутом или пряником — всё-таки сделать то, что необходимо.

Для этого подойдут игровые механики. Допустим, надо подготовить квартальный отчёт. Мы можем использовать кнут: установить напоминалку о том, что осталось всего два дня до дедлайна, и штрафовать за опоздания. А можно использовать пряник: сделать так, что, удовлетворяя свои собственные потребности, сотрудник автоматически выполнит бизнес-задачу. Условно говоря, чтобы накормить своего тамагочи, он должен будет отправить квартальный отчёт. У нас был такой проект: несколько отделов логистики компании соревновались друг с другом в числе обработанных заказов, а в конце месяца победитель получал бонус. Но в течение месяца никто не знал, кто впереди. Когда мы вытащили соревнование в интерфейс

так, что сотрудник в реальном времени видел, что от его заказа зависит судьба бонуса всего отдела, производительность труда заметно повысилась.

Наша цель — сделать интерфейс эффективным. Эти задачи часто идут рука об руку. Но бывают и исключения. Например, если мы знаем, что цикл жизни продукта — полгода, а затраты на новую версию не окупятся в первые месяцы его работы, мы просто исправим мелкие ошибки в старом интерфейсе и не будем ничего кардинально менять.

Отличный пример из прошлого: известно, что танки Т-34 были неудобными. Когда стали решать, стоит ли в них что-то менять, то поняли, что делать это бессмысленно. Среднее время участия танка в бою — 11 минут, и за это время танкисты не успевают устать. В остальное время экипаж редко находится внутри танка: он едет на броне. Задача сделать всё максимально удобным стоит не всегда.

Три основных показателя юзабилити: эффективность, продуктивность и удовлетворённость. Эффективность — процент успешно завершённых сценариев. Продуктивность — сколько времени, внимания пользователь потратил на это. Субъективная удовлетворённость определяет, насколько комфортен этот процесс для него. Первые два фактора можно измерить напрямую, а последний пока только косвенно — через анкеты.

Но уже сейчас игровая индустрия активно внедряет новые технологии измерения реакций пользователей. Там уже строят интерфейсы на основе оценки психофизиологического состояния игрока. Если система понимает, что ты нервничаешь, твой прицел в стрелялке становится больше. В Mail.ru есть лаборатория, где отслеживают различные параметры игрока: частоту пульса, дыхания, кожно-гальваническое сопротивление и другое. Пока это делают специальные аппаратно-программные комплексы. Но думаю, в скором времени такие измерения могут стать доступными при помощи веб-камеры, которая будет замерять частоту пульса по зрачкам. Этот тренд распространится и за пределы индустрии игр. Система будет знать многое о вашем состоянии и на основе этого взаимодействовать с вами. Например, если станет ясно, что вы сосредоточены на работе и вас лучше не беспокоить, она не будет показывать вам уведомления о письмах со стандартным и низким приоритетом.

Одно из главных правил хорошего интерфейса: нужно ориентироваться на ментальную модель пользователя, а не на реальную структуру системы.

Ментальная модель — это то, как он представляет себе систему. Чаще всего он воспринимает её упрощённо. Например, когда мы говорим по телефону, мы представляем, как радиоволны бегут от одного к другому. В реальности всё сложнее, но для разговора нам знать об этом не нужно. Люди ожидают от системы определённого поведения, и на это стоит ориентироваться.

Важно делать интерфейс для реальных людей. Часто заказчики делают продукт под самих себя. Мы тестировали интерфейс одного клиент-банка и заметили, что 80% пользователей не могли выполнить задачу «отправить документ бенефициару», потому что не знали слова «бенефициар». Когда мы рассказали об этом заказчикам, они ответили, что только идиоты не знают этого слова. То есть они не могли представить, что их пользователи могут быть абсолютно не похожи на них.

Tcnm концепция MetroUI, которая уловила важный тренд: люди перестали бояться быть цифровыми. Раньше многим интерфейсам была свойственна метафоричность. Мы с помощью объектов на экране изображали объекты из реальной жизни — только так можно было дать представление об их функциях. Люди видели кнопку и понимали, что на неё можно нажать, видели корзинку и понимали, что в неё можно выбросить ненужные документы. Сейчас настал момент, когда можно отказаться от искусственной связи с реальным миром и делать чисто цифровые интерфейсы. Дети знакомятся с компьютером раньше, чем с корзиной для бумаг.

Проектировать нужно не интерфейс, а сервис, где продукт является лишь одной из точек контакта с пользователем. Это может быть и реклама на радио, и офис, и сайт. Все они имеют смысл, если помогают пользователю решить его проблему, удовлетворить потребность. Сам по себе интерфейс никому не нужен, потому что он самостоятельно не решает никакую задачу. Так же, как и продукт. Сайт банка без банка — бред. Я заполняю анкету на сайте, и мне предлагают после этого прийти в отделение — это вроде бы нормально и современно. Но на деле даже этот момент — точка разрыва сервиса с пользователем. Чтобы не оставлять его предоставленным самому себе, можно предложить выбрать ближайшее к нему отделение, назначить удобное время визита или напомнить о документах, которые нужно принести. Не говоря уж о постоянном ведении клиента с использованием смартфона.

Ещё нужно учитывать, что люди используют несколько устройств для работы с одним и тем же контентом. Сначала ищут информацию на телефоне, а потом читают об этом подробнее на компьютере. В скором времени можно будет

параллельно с просмотром футбольного матча на ТВ смотреть составы команд и повтор голов на вашем планшете.

Пользователей часто представляют людьми, которых нет в природе: счастливыми, здоровыми, подкованными в технологиях. В первых интерфейсах платёжных автоматов вместо кнопок в качестве активных элементов использовали ссылки с текстом. И (внезапно!) оказалось, что у половины населения страны они не вызывают никаких ассоциаций с функциональными элементами, потому что она не пользуется интернетом. Чтобы сделать хороший интерфейс, надо чётко понимать, кто твой пользователь, навыками работы с какими программами он обладает, к каким паттернам поведения он привык. Знаете, что он знаком с клавиатурными сокращениями, — добавляйте эту функцию в свой интерфейс. Нет — водите его за курсор от кнопки к кнопке.

Даже если у вас нет ресурсов на обширное исследование аудитории, всё равно опишите своего пользователя и его потребности. Пусть даже с потолка. Это позволит критически взглянуть на планы и функционал, сузит фронт работ и задаст вектор развития сервиса. Делать хорошо «вообще для всех» — значит не делать хорошо ни для кого.

1.2 Основы построения интерфейсов

Для обеспечения успешной работы пользователя от дизайнера интерфейса требуется соблюдать баланс между вышеперечисленными факторами на протяжении всего жизненного цикла разработки приложения. Это достигается последовательной и тщательной проработкой деталей интерактивного взаимодействия на каждом из этапов разработки пользовательского интерфейса, включающих:

1. Проектирование
    - Функциональные требования: определение цели разработки и исходных требований.
    - Анализ пользователей: определение потребностей пользователей, разработка сценариев, оценка соответствия сценариев ожиданиям пользователей.
    - Концептуальное проектирование: моделирование процесса, для которого разрабатывается приложение.

- Логическое проектирование: определение информационных потоков в приложении.
- Физическое проектирование: выбор платформы, на которой будет реализован проект и средств разработки.

2. Реализация
   - Прототипирование: разработка бумажных и/или интерактивных макетов экранных форм.
   - Конструирование: создание приложения с учетом возможности изменения его дизайна.
3. Тестирование
   - Юзабилити-тестирование: тестирование приложения различными пользователями, в т.ч. и пользователями с ограниченными возможностями (Accessibility testing).

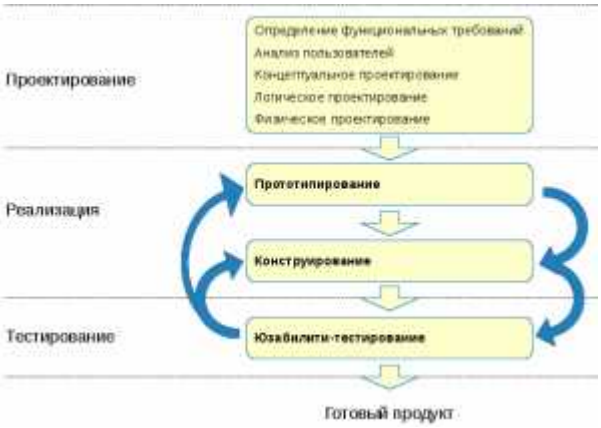


Рисунок 2 - Этапы разработки пользовательского интерфейса

Как и разработка приложения в целом, создание пользовательского интерфейса для него — процесс итеративный. Маловероятно, что такие этапы, как прототипирование, конструирование и тестирование интерфейса могут быть завершены за один проход. Поэтому, если в результате юзабилити-тестирования выявлены недоработки, то они, если это возможно, устраняются путем повторного конструирования, либо разрабатывается новый прототип интерфейса.

Когда говорят о научных основах проектирования пользовательских интерфейсов, в первую очередь упоминают термин HCI. HCI — это аббревиатура английского Human-Computer Interaction, что переводится как "взаимодействие человека и компьютера". На Западе HCI — это целая профессия, ей обучают в университетах, выдавая дипломы "Специалист по HCI". Издается много журналов по этой теме, существует большое количество Web-сайтов. В России, к сожалению,

эта наука не пользуется особой популярностью например, у нас настоящих специалистов по HCI можно буквально пересчитать по пальцам одной руки.

Как легко догадаться по названию, составными частями HCI являются:

- человек (пользователь)
- компьютер
- их взаимодействие.

Пользовательский интерфейс (англ. user interface, UI) является своеобразным коммуникационным каналом, по которому осуществляется взаимодействие пользователя и компьютера.

Лучший пользовательский интерфейс — это такой интерфейс, которому пользователь не должен уделять много внимания, почти не замечать его. Пользователь просто работает, вместо того, чтобы размышлять, какую кнопку нажать или где щелкнуть мышью. Такой интерфейс называют прозрачным — пользователь как бы смотрит сквозь него на свою работу.

Чтобы создать эффективный интерфейс, который делал бы работу с программой приятной, нужно понимать, какие задачи будут решать пользователи с помощью данной программы и какие требования к интерфейсу могут возникнуть у пользователей. Это сделать гораздо легче, если вы используете свою программу для собственных нужд, ведь в данном случае вы являетесь не только разработчиком, но и пользователем программы, смотрите на нее глазами ее аудитории.

Огромную роль играет интуиция — если разработчик сам терпеть не может некрасивые и неудобные интерфейсы, то при создании собственной программы он будет чувствовать, где и какой именно элемент нужно убрать или добавить. Необходимо иметь художественный вкус, чтобы понимать, что именно придаст интерфейсу красоту и привлекательность.

Западные исследователи в области HCI сформулировали основные принципы проектирования пользовательских интерфейсов компьютерных программ. Как и в любой другой науке, существует довольно много различных методик и классификаций, которые можно найти в книгах по HCI, выпушенных за рубежом, а также на иностранных Web-сайтах.

Если говорить о самых общих принципах проектирования пользовательских интерфейсов, то можно назвать три основных положения:

1. Программа должна помогать выполнить задачу, а не становиться этой задачей.

2. При работе с программой пользователь не должен ощущать себя дураком.

3. Программа должна работать так, чтобы пользователь не считал компьютер дураком[2].

Довольно эмоциональные формулировки, но, тем не менее, поразительно верные.

Первый принцип — это уже упоминавшаяся выше прозрачность интерфейса. Интерфейс должен быть легким для освоения и не создавать перед пользователем преграду, которую он должен будет преодолеть, чтобы приступить к работе.

Второй принцип часто нарушают те авторы программ, которые слишком недооценивают умственные способности пользователей. В глазах таких разработчиков пользователи видятся толпой этаких бестолковых болванов, в лучшем случае — беспомощных и нерадивых созданий, не способных разобраться в самых элементарных ситуациях. Это обусловлено разными причинами.

Во-первых, традиционным слегка высокомерным отношением программистов к простым пользователям. Это еще можно было понять в восьмидесятых и начале девяностых годов XX века, когда обычные персональные компьютеры не имели доступных широкой аудитории программных и аппаратных средств для построения привлекательных графических интерфейсов и работы с ними. Самой распространенной операционной системой в то время была MS DOS, основанная на интерфейсе командной строки. Поэтому эффективно работать с персональным компьютером могли люди только с довольно серьезной подготовкой. Кроме того, парк "персоналок" был относительно невелик даже в США, не говоря уже об остальных странах, и, как следствие, число пользователей компьютеров было небольшим.

Сегодня же такой пренебрежительный взгляд на пользователя явно неуместен. Работа с персональным компьютером предполагает относительно не большую начальную подготовку пользователя: интерфейсы компьютерных программ, в первую очередь операционной системы Windows, являющейся законодателем мод в индустрии массового программного обеспечения, становятся все проще и доступнее для понимания людей. Да и число компьютеров в мире сегодня в

несколько раз больше, чем десять лет назад.

Вторая причина слишком большой недоверчивости программистов к познаниям и квалификации пользователей - чрезмерное увлечение построением так называемой "защиты от дурака". Дело в том, что классические учебные курсы по программированию учат, что большинство ошибок в работе программы вызываются не дефектами исходного кода или программного окружения, а действиями пользователя — например, вводом данных неправильного формата (допустим, текста вместо цифр). Поэтому программист при разработке приложения должен написать функции по проверке результатов как можно большего числа действий пользователя и предусмотреть максимальное количество вариантов развития событий. Происходит довольно обычная вещь: то, что задумывалось как решение проблемы, само начинает создавать проблемы.

И, наконец, *третья причина* во многом обусловлена поведением самих пользователей. Часто при возникновении малейших затруднений при работе с программой пользователь тут же обращается в службу технической поддержки, не удосужившись даже взглянуть на справочную систему продукта, посмотреть секцию "*Ответы на частые вопросы*" на Web-сайте программы или даже просто чуть-чуть подумать! Отчасти тут вина самих авторов программ.

Несмотря на стремительное развитие информационных технологий, многие компьютерные программы все еще имеют примитивный искусственный интеллект. Они прерывают работу пользователя глупыми вопросами и выводят на экран бессмысленные сообщения, повергая его в недоумение в самых простых ситуациях.

2. ПРИНЦИП ПОСТРОЕНИЯ ИНТЕРФЕЙС ПРОГРАММЫ

Разработка интерфейса обычно начинается с определения задачи или набора задач, для которых продукт предназначен

Простое должно оставаться простым. Не усложняйте интерфейсы. Постоянно думайте о том, как сделать интерфейс проще и понятнее.

Пользователи не задумываются над тем, как устроена программа. Все, что они видят — это интерфейс. Поэтому, с точки зрения потребителя именно интерфейс является конечным продуктом.

Интерфейс должен быть ориентированным на человека, т.е. отвечать нуждам человека и учитывать его слабости. Нужно постоянно думать о том, с какими трудностями может столкнуться пользователь.

Думайте о поведении и привычках пользователей. Не меняйте хорошо известные всем ЭИ на неожиданные, а новые делайте интуитивно понятными.

2.1 Дизайн ЭИ

На самом деле, дизайн ЭИ — тема отдельной статьи. Тут нужно учитывать все: начиная от цвета, формы, пропорций, заканчивая когнитивной психологией. Однако, несколько принципов все же стоит отметить:

Цвет. Цвета делятся на теплые(желтый, оранжевый, красный), холодные(синий, зеленый), нейтральные(серый). Обычно для ЭИ используют теплые цвета. Это как раз связано с психологией восприятия. Стоит отметить, что мнение о цвете — очень субъективно и может меняться даже от настроения пользователя.

Форма. В большинстве случаев — прямоугольник со скругленными углами. Или круг. Полностью прямоугольные ЭИ, лично мне нравятся меньше. Возможно из-за своей «остроты». Опять же, форма как и цвет достаточно субъективна.

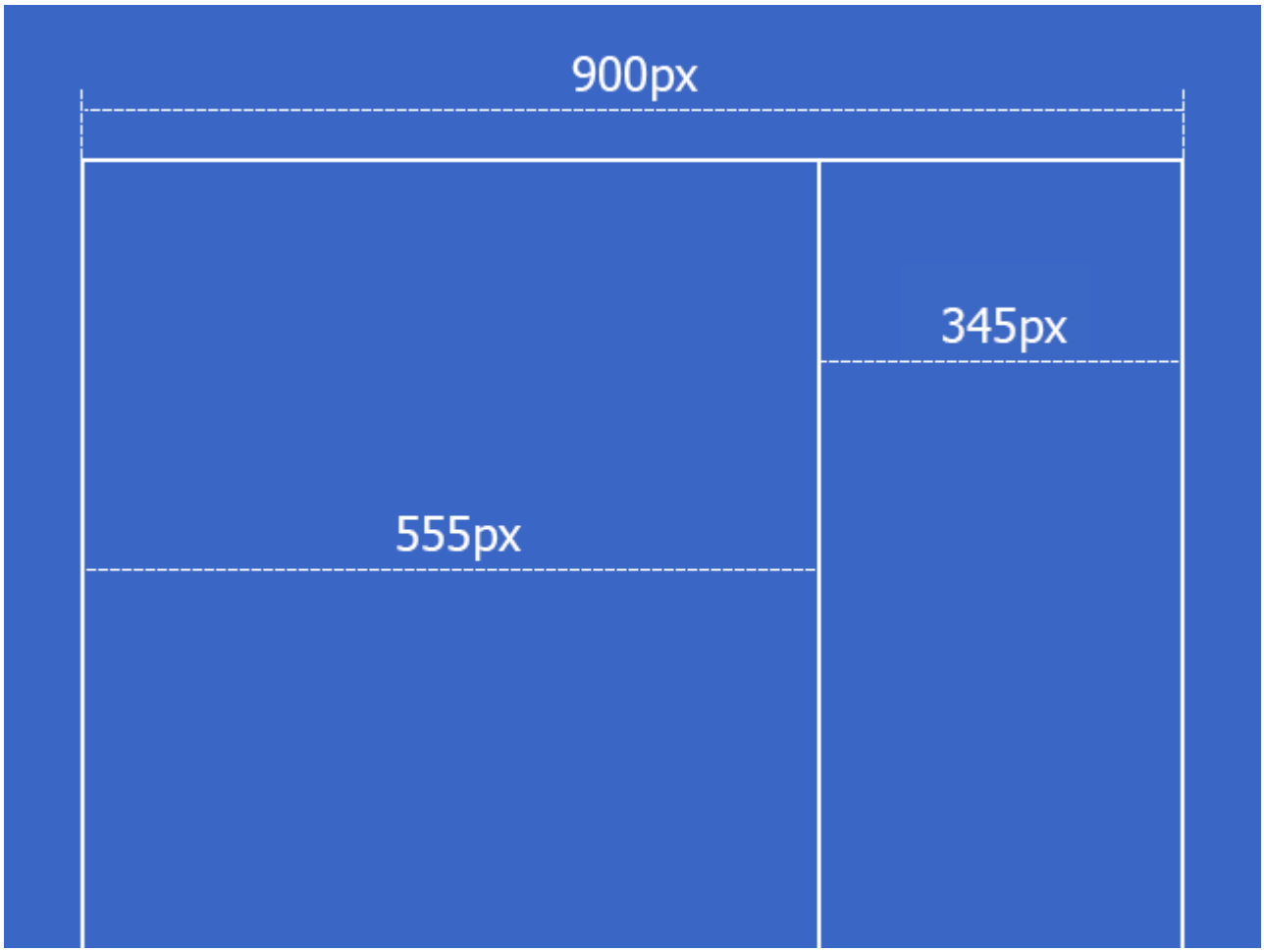
Основные ЭИ(часто используемые) должны быть выделены. Например размером или цветом.

Иконки в программе должны быть очевидными. Если нет — подписывайте. Ведь, по сути дела, вместо того чтобы объяснять, пиктограммы зачастую сами требуют для себя объяснений.

Старайтесь не делать слишком маленькие элементы — по ним очень трудно попасть.

2.2 Расположение ЭИ на экране

Есть утверждение, что визуальная привлекательность основана на пропорциях. Суть в том, что весь отрезок относится к большей его части так, как большая часть, относится к меньшей. Например, общая ширина сайта 900px, делим 900 на 1.62, получаем ~555px, это ширина блока с контентом. Теперь от 900 отнимаем 555 и получаем 345px. Это ширина меньшей части[3]:



Перед расположением, ЭИ следует упорядочить (сгруппировать) по значимости. Т.е. определить, какие наиболее важны, а какие — менее.

Обычно(но не обязательно), элементы размещаются в следующей градации: слева направо, сверху вниз. Слева в верху самые значимые элементы, справа внизу — менее. Это связано с порядком чтения текста. В случае с сенсорными экранами, самые важные элементы, располагаются в области действия больших пальцев рук.

Необходимо учитывать привычки пользователя. Например, если в Windows кнопка закрыть находится в правом верхнем углу, то программе аналогичную кнопку необходимо расположить там же. Т.е. интерфейс должен иметь как можно больше аналогий, с известными пользователю вещами.

Размещайте ЭИ поближе там, где большую часть времени находится курсор пользователя. Что бы ему не пришлось перемещать курсор, например, от одного конца экрана к другому.

Соблюдайте пропорции

Элемент интерфейса можно считать видимым, если он либо в данный момент доступен для органов восприятия человека, либо он был настолько недавно воспринят, что еще не успел выйти из кратковременной памяти. Для нормальной работы интерфейса, должны быть видимы только необходимые вещи — те, что идентифицируют части работающих систем, и те, что отображают способ, которым пользователь может взаимодействовать с устройством.

Делайте отступы между ЭИ равными или кратными друг-другу.

Пользователи привыкают. Например, при удалении файла, появляется окно с подтверждением: «Да» или «Нет». Со временем, пользователь перестает читать предупреждение и по привычке нажимает «Да». Поэтому диалоговое окно, которое было призвано обеспечить безопасность, абсолютно не выполняет своей роли. Следовательно, необходимо дать пользователю возможность отменять, сделанные им действия[4].

Если вы даете пользователю информацию, которую он должен куда-то ввести или как-то обработать, то информация должна оставаться на экране до того момента, пока человек ее не обработает. Иначе он может просто забыть.

Избегайте двусмысленности. Например, на фонарике есть одна кнопка. По нажатию фонарик включается, нажали еще раз — выключился. Если в фонарике перегорела лампочка, то при нажатии на кнопку не понятно, включаем мы его или нет. Поэтому, вместо одной кнопки выключателя, лучше использовать переключатель(например, checkbox с двумя позициями: «вкл.» и «выкл.»). За исключением случаев, когда состояние задачи, очевидно.

Такой переключатель напрямую отражает состояние ЭИ.

Делайте монотонные интерфейсы. Монотонный интерфейс — это интерфейс, в котором какое-то действие, можно сделать только одним способом. Такой подход обеспечит быструю привыкаемость к программе и автоматизацию действий.

Не стоит делать адаптивные интерфейсы, которые изменяются со временем. Так как для выполнения какой-то задачи, лучше изучать только один интерфейс, а не несколько. Пример — стартовая страница браузера Chrome.

Если задержки в процессе выполнения программы неизбежны или действие производимое пользователем очень значимо, важно, чтобы в интерфейсе была предусмотрена сообщающая о них обратная связь. Например, можно использовать индикатор хода выполнения задачи (status bar).

ЭИ должны отвечать. Если пользователь произвел клик, то ЭИ должен как-то отозваться, чтобы человек понял, что клик произошел.

ЗАКЛЮЧЕНИЕ

Интерфейс представляет собой систему связи между различными узлами и блоками сложного оборудования, а также между техникой и пользователем. Он выражается в логической (системы представления информации) и физической (характеристики информационных сигналов) форме.

Так, логически компьютерные интерфейсы представляют собой сложные математические системы, основанные на понятиях Булевой алгебры, а физически – это совокупность чипов и других электронных деталей, медных проводов и импульсов электрического тока.

Интерфейс пользователя – эта та часть программы, которая находится у всех на виду. Некоторые программисты склонны оставлять дизайн интерфейса пользователя на потом, считая, что реальное достоинство приложения – его программный ко. который и требует большего внимания. Однако часто возникает недовольство пользователей из-за неудачно подобранных шрифтов, непонятного

содержимого экрана и скорости его прорисовывания, поэтому работу над интерфейсом также нужно воспринимать серьезно. Пользователь не видит программного кода, зато интерфейс (хороший или плохой) всегда перед ним.

Хотя ни одно ухищрение не гарантирует создания удачного интерфейса пользователя, плохой интерфейс гарантирует отсутствие пользователей вашей программы. Однако при стремительном появлении новшеств, в сфере пользовательских интерфейсов, понятие “хорошего” интерфейса очень быстро изменяется.

Так и интерфейс пользователя программ будет “эволюционировать” по мере того как индустрия будет устанавливать новые стандарты, и вы, в свою очередь, должны быть всегда в курсе, как в наибольшей степени удовлетворить ожидания пользователя.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Абрамов К.Ю. Информатика, М., 2011. – 314 с.
2. Агафонов А.А. Программирование, 2014. – 312 с.
3. Бальмонт П.О. Основы информатики. М., 2012. – 412 с.
4. Ваучер К.У. Информатика, СПб., 2013. – 514 с.
5. Голдман М. Информатика. М.: ФСПИ «Тренды», 2014. – 455 с.
6. Голдман М. Основы программирования. М.: ЭПИцентр – Интеграл-Информ, 2013. – 489 с.
7. Демидов У.В. Программирование, М., 2011. – 289 с.
8. Интерфейс: новые направления в проектировании компьютерных систем. М.: Экономика, 2014. – 118 с.
9. Интерфейсы/ Под ред. А.Б. Чубайса. М., 2014. – 331 с.
10. Манин Р.П. Информатика, СПб., 2014. – 457 с.
11. Носов П.А., Интерфейсы , М., 2015. – 411 с.
12. Нуриков А.В. Информатика, СПб., 2013. – 145 с.
13. Об интерфейсе. Основы проектирования взаимодействия: Учеб. пособие. М.: ИЭПП, 2011. – 436 с.
14. Олейников М.С. Информатика: Учеб. пособие. М., 2015. - 277 с.
15. Хеллман А. Интерфейс: Учеб. пособие. 2012. – 258 с.
16. Явлинский Г.А. Информатика: Учеб. пособие. М.: ГУ – ВШЭ, 2014. – 319 с.

1. 1. Интерфейсы/ Под ред. А.Б. Чубайса. М., 2014. – 331 с.

↑

2. 1. Интерфейсы/ Под ред. А.Б. Чубайса. М., 2014. – 331 с.

↑

3. Олейников М.С. Информатика: Учеб. пособие. М., 2015. - 277 с. ↑

4. 1. Явлинский Г.А. Информатика: Учеб. пособие. М.: ГУ – ВШЭ, 2014. – 319 с.

↑