**09.03.2020**

**Проведение предпроектных исследований. Разработка технического задания.**

Осуществляется с целью параметризации проекта создания ИС. Сначала выявляются все материальные, финансовые людские и временные ресурсы для выполнения необходимых проектных работ. Это этап сбора необходимых данных и их анализа.

На основе полученных данных формируется Техническое задание (ТЗ) – третий этап проектирования (при необходимости – технико-экономическое обоснование проекта – ТЭО), и собственно проект (четвёртый этап проектирования).

***Техническое задание***

*Техническое задание* (ТЗ) – стадия проектирования ИС. Оно формируется по результатам проведённого предпроектного исследования и включает разработку и утверждение Технического задания на создание ИС.

Напомним, что разработчики к нему могут и не приступить, если в результате предпрпоектногоф исследования было выявлено, что Заказчику следует отказаться от использования ИС.

В зависимости от условий создания системы возможны различные совмещения функций заказчика, разработчика, поставщика и других организаций, участвующих в работах по созданию ИС.

В общем случае *содержание технического задания* включает следующие разделы:

1. Введение;
2. Основание для разработки;
3. Назначение разработки;
4. Технические требования к программе или программному изделию;
5. Технико-экономические показатели;
6. Стадии и этапы разработки;
7. Порядок контроля и приёмки;
8. Приложения.

В зависимости от вида, назначения, специфических особенностей объекта автоматизации и условий функционирования системы допускается оформлять разделы ТЗ в виде приложений, вводить дополнительные, исключать или объединять подразделы ТЗ.

В состав Технического задания входят следующие сведения:

* наименование системы;
* данные о Заказчике и Исполнителях;
* конфигурация технических средств;
* стоимостные данные;
* описание АИС, включающее поддерживаемые данным средством процессы создания и сопровождения АИС, программную среду, функции, входные/выходные данные и область применения.

В состав ТЗ на ИС включают приложения, содержащие:

* Расчёт ожидаемой эффективности системы;
* Оценку научно-технического уровня системы;
* Использованные при разработке ТЗ методические и наиболее важные информационные материалы из состава документов, указанных в разделе «Источники разработки».

**10.03.2020**

**Проведение тестирования алгоритма и программного продукта**

***Тестирование и его результаты***

Тестирование – ряд мероприятий, связанных с различного рода испытаниями объекта тестирования с целью установления соответствия или несоответствия его характеристик определенным требованиям и выявления дефектов. Дефектами, в свою очередь, могут быть как ошибки в работе, так и неприемлемое качество функционирования в определенных условиях эксплуатации.

Результаты тестирования - это список выявленных несоответствий и дефектов, но, как правило, без указания их причин, хотя, опытный специалист по качеству продукции в состоянии провести первичную диагностику некорректного поведения объекта и сообщить диагноз его разработчикам.

Тестирование программного продукта проводят как на этапах его создания, так и на этапах технического сопровождения и дальнейшего развития. Виды тестирования программных продуктов и их краткое описание я постараюсь представить в этом материале.

***Тестирование на этапе создания программного продукта***

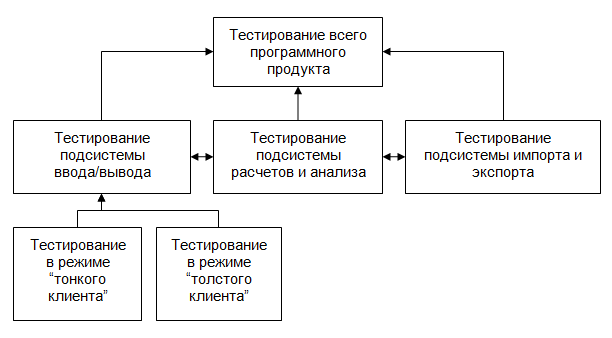
Виды тестирования на этапе [создания программного продукта](http://codingcraft.ru/programming_for_beginners.php) классифицируют по преследуемым этим процессом целям. Функциональное тестирование - проверка корректного выполнения программой заложенных в нее функций. Нефункциональное тестирование - все прочие виды испытаний, такие как тестирование производительности в различных режимах эксплуатации, тестирование эргономики пользовательского интерфейса, тестирование отказоустойчивости и т.д.

***Функциональное тестирование***

Для проведения функционального тестирования персоналом отдела технического контроля разрабатывается документ программа и методика испытаний функционала приложения (ПМИ). Документ ПМИ содержит перечень сценариев тестирования программного продукта (test cases) с подробным описанием шагов. Каждый шаг сценария тестирования характеризуется действиями пользователя (специалиста по тестированию) и ожидаемыми результатами – ответной реакции программы на эти действия. Программа и методика испытаний обязана имитировать эксплуатацию программного продукта в реальном режиме. Это означает, что сценарий тестирования должен быть построен на основе анализа операций, которые будут выполнять будущие пользователи системы, а не быть искусственно составленной последовательностью понятных только разработчику манипуляций. Функциональное тестирование может проводиться на различных уровнях тестирования, перечень которых зависит от сложности приложения:

* **Компонентное (модульное) тестирование.** Тестирование отдельных компонентов программного продукта, сфокусированное на их специфике, назначении и функциональных особенностях.
* **Интеграционное тестирование.** Данный вид тестирования проводится после компонентного тестирования и направлен на выявление дефектов взаимодействия различных подсистем на уровне потоков управления и обмена данными.

Каждый компонент системы может быть рассмотрен, как отдельная подсистема, и внутри этой подсистемы могут быть выделены ее компоненты. В этом случае для таких подсистем отдельно проводятся компонентное и интеграционное тестирование. Таким образом, для сложных программных продуктов получаем иерархическую структуру процесса тестирования, на каждом уровне которой объектом тестирования является определенная часть программного комплекса.

  
Рисунок 1 - Пример иерархической структуры процесса тестирования программного продукта

Выделение уровней может происходить по принципу общей функциональности (подсистема ввода/вывода данных, подсистема расчетов и аналитики, подсистема хранения данных и т.п.), по принадлежности к конкретной части проектного решения (сервер, клиент, посредник), по используемым технологиям, либо по всему сразу. В этом случае тестирование может проводиться снизу вверх, и при каждом переходе на более высокий уровень, протестированные ранее части программы уже выступают в качестве компонентов подсистемы более высокого порядка. Бывает, что проводят тестирование сверху вниз, начиная от испытаний, связанных с общим поведением программы и заканчивая деталями. В этой ситуации может оказаться так, что на высоких уровнях абстракции реализация деталей еще отсутствует и заменяется различного рода имитаторами – функциональными заглушками, которые только со временем, при переходе на более низкие уровни заменяются реальными функциональными компонентами. Такой процесс на ранних этапах является ни чем иным как тестированием прототипа программного продукта.

### ***Нефункциональное тестирование***

Нефункциональное тестирование, как следует из самого названия, в первую очередь оценивает второстепенные качества функционирования программного продукта. Второстепенные не в том смысле, что эти качества программного продукта менее важны, чем корректное выполнение им своих основных функций, а в том, что они никому не интересны, если продукт, в принципе, не выполняет то, что должен. Кому, например, интересны результаты замеров производительности подсистемы расчетов, если сами результаты этих расчетов не верны? Для кого имеет значение удобство пользовательского интерфейса программы, если его работа не приводит к ожидаемым от него результатам? Ну а насчет важности нефункционального тестирования можно сказать так: иногда его результаты имеют более серьезный вес в общей оценке работы программного продукта, чем результаты функционального тестирования. Например, для системы, работающей в реальном времени куда как более важно успевать взаимодействовать со всеми наблюдаемыми внешними объектами, чем корректно, с точки зрения функциональных требований, обрабатывать события от конкретного источника. Повторюсь, что это вовсе не означает, что можно некорректно взаимодействовать с отдельно-взятым объектом в угоду производительности.

#### **Тестирование производительности**

Достаточно интересные виды тестирования с точки зрения условий их проведения и оценки результатов. В ходе этапа тестирования производительности в первую очередь проводят **нагрузочное тестирование**, целью которого является проверка, будет ли система адекватно реагировать на внешние воздействия в режиме, близком к режиму реальной эксплуатации. Адекватная реакция – это время отклика в пределах, установленных требованиями к программному продукту. Внешние воздействия (нагрузка) – это, чаще всего, запросы пользователей. Ну а режим реальной эксплуатации подразумевает то, что система развернута на соответствующем требованиям аппаратном обеспечении, а самих пользователей примерно столько, сколько и планировалось. Кроме нагрузочного тестирования проводят испытания в условиях минимальных аппаратных средств и максимальной нагрузки – **стрессовое тестирование**, а также, испытания в условиях предельных объемов обрабатываемой информации **– объемное тестирование**. Выделяют еще один вид тестирования: **тестирование стабильности и надежности**, которое включает в себя не только длительное испытание программного продукта в нормальных условиях, но и способность его возвращаться в нормальный режим функционирования после непродолжительных периодов стрессовых нагрузок.

#### **Прочие виды нефункционального тестирования**

К прочим видам тестирования относят **тестирование процесса установки** или развертывания программного продукта, **тестирование эргономики** пользовательского интерфейса (не корректной его работы, а именно удобства использования), **тестирование способности системы к восстановлению нормальной работы** после серьезных сбоев, вызванных отказами аппаратного или системного программного обеспечения. Отдельно проводят **испытания системы на различных конфигурациях**, если требованиями они предусмотрены. Конфигурации могут отличаться вплоть до операционных систем серверной и клиентской частей программного комплекса. Тестирование отдельной конфигурации может сводиться к проведению для нее всего комплекса испытаний программного продукта.

## Тестирование на этапе сопровождения программного продукта

### ***Регрессионное тестирование***

Регрессионное тестирование проводят по результатам исправления выявленных на этапе эксплуатации программного продукта ошибок и дефектов. Также, к этому виду тестирования относят испытания программного продукта после внесения в него незначительных изменений, которые не должны влиять на общую функциональность, а вызваны такими обстоятельствами, как необходимость перехода на новую версию операционной системы или новый набор прикладных библиотек стороннего производителя. Цель регрессионного тестирование проста: доказать, что “ничего не сломалось”, и что программный продукт по-прежнему соответствует всем заявленным ранее требованиям.

### ***Предварительное тестирование новой версии программного продукта***

Комплекс предварительных мероприятий, направленный на то, чтобы быстро показать жизнеспособность новой версии программного продукта или отправить ее на доработку, выявив в максимально короткие сроки наиболее серьезные дефекты. Ключевое слово в определении – быстро. Подобный вид тестирования еще называют **дымовым тестированием** по аналогии с испытаниями печниками новых печей: если дым не повалил оттуда, откуда не положено, значит все в порядке. Если предварительное тестирование заканчивается успешно, новую версию программного продукта отправляют на более детальное “обследование”, которое включает функциональное и нефункциональное тестирование.

**11.03.2020**

**Отладка программного обеспечения**

Отладка программы — один их самых сложных этапов разработки программного обеспечения, требующий глубокого знания:

* специфики управления используемыми техническими средствами,
* операционной системы,
* среды и языка программирования,
* реализуемых процессов,
* природы и специфики различных ошибок,
* методик отладки и соответствующих программных средств.

Отладка - это процесс локализации и исправления ошибок, обнаруженных при тестировании программного обеспечения. Локализацией называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Исправления ошибки необходимо определить ее причину, т. е. определить оператор или фрагмент, содержащие ошибку. Причины ошибок могут быть как очевидны, так и очень глубоко скрыты.

В соответствии с этапом обработки, на котором проявляются ошибки, различают: синтаксические ошибки - ошибки, фиксируемые компилятором при выполнении синтаксического и частично семантического анализа программы; ошибки компоновки - ошибки, обнаруженные компоновщиком (редактором связей) при объединении модулей программы; ошибки выполнения - ошибки, обнаруженные операционной системой, аппаратными средствами или пользователем при выполнении программы.

***Синтаксические ошибки.***

Синтаксические ошибки относят к группе самых простых, так как синтаксис языка, как правило, строго формализован, и ошибки сопровождаются развернутым комментарием с указанием ее местоположения. Определение причин таких ошибок, как правило, труда не составляет, и даже при нечетком знании правил языка за несколько прогонов удается удалить все ошибки данного типа.

***Ошибки выполнения.***

К самой непредсказуемой группе относятся ошибки выполнения. Прежде всего они могут иметь разную природу, и соответственно по-разному проявляться. Часть ошибок обнаруживается и документируется операционной системой. Выделяют четыре способа проявления таких ошибок: • появление сообщения об ошибке, зафиксированной схемами контроля выполнения машинных команд, например, переполнении разрядной сетки, ситуации «деление на ноль», нарушении адресации и т. п.; •появление сообщения об ошибке, обнаруженной операционной системой, например, нарушении защиты памяти, попытке записи на устройства, защищенные от записи, отсутствии файла с заданным именем и т. п.; •«зависание» компьютера, как простое, когда удается завершить программу без перезагрузки операционной системы, так и «тяжелое», когда для продолжения работы необходима перезагрузка; •несовпадение полученных результатов с ожидаемыми.

***Методы отладки программного обеспечения.***

Отладка программы в любом случае предполагает обдумывание и логическое осмысление всей имеющейся информации об ошибке. Большинство ошибок можно обнаружить по косвенным признакам посредством тщательного анализа текстов программ и результатов тестирования без получения дополнительной информации. При этом используют различные методы:

1) ручного тестирования;

2) индукции;

3) дедукции;

4) обратного прослеживания.

***Метод ручного тестирования.***

Это - самый простой и естественный способ данной группы. При обнаружении ошибки необходимо выполнить тестируемую программу вручную, используя тестовый набор, при работе с которым была обнаружена ошибка. Метод очень эффективен, но не применим для больших программ, программ со сложными вычислениями и в тех случаях, когда ошибка связана с неверным представлением программиста о выполнении некоторых операций. Данный метод часто используют как составную часть других методов отладки.

***Индукции***

Метод основан на тщательном анализе симптомов ошибки, которые могут проявляться как неверные результаты вычислений или как сообщение об ошибке.

***Дедукции***

По методу дедукции вначале формируют множество причин, которые могли бы вызвать данное проявление ошибки. Затем анализируя причины, исключают те, которые противоречат имеющимся данным. Если все причины исключены, то следует выполнить дополнительное тестирование исследуемого фрагмента. В противном случае наиболее вероятную гипотезу пытаются доказать. Если гипотеза объясняет полученные признаки ошибки, то ошибка найдена, иначе - проверяют следующую причину.

***Метод обратного прослеживания.***

Для небольших программ эффективно применение метода обратного прослеживания. Начинают с точки вывода неправильного результата. Для этой точки строится гипотеза о значениях основных переменных, которые могли бы привести к получению имеющегося результата. Далее, исходя из этой гипотезы, делают предложения о значениях переменных в предыдущей точке. Процесс продолжают, пока не обнаружат причину ошибки.

**12.03.2020**

**Составление описания на программный продукт**

Требования к структуре описания программного обеспечения по ГОСТ 34 устанавливаются [РД 50-34.698-90](http://technicaldocs.ru/%D0%B3%D0%BE%D1%81%D1%8234/%D0%BD%D0%BF%D0%B0/%D1%80%D0%B450-34.698-90). В общем случае документ должен состоять из следующих разделов:

1. Общее описание системы

Документ содержит разделы:

* назначение системы;
* описание системы;
* описание взаимосвязей АС с другими системами;
* описание подсистем (при необходимости).
  1. В разделе «Назначение системы» указывают:
* вид деятельности, для автоматизации которой предназначена система;
* перечень объектов автоматизации, на которых используется система;
* перечень функций, реализуемых системой.
  1. В разделе «Описание системы» указывают:
* структуру системы и назначение ее частей;
* сведения об АС в целом и ее частях, необходимые для обеспечения эксплуатации системы;
* описание функционирования системы и ее частей.
  1. В разделе «Описание взаимосвязей АС с другими системами» указывают:
* перечень систем, с которыми связана данная АС;
* описание связей между системами;
* описание регламента связей;
* описание взаимосвязей АС с подразделениями объекта автоматизации.
  1. В разделе «Описание подсистем» указывают:
* структуру подсистем и назначение ее частей;
* сведения об подсистемах и их частях, необходимые для обеспечения их функционирования;
* описание функционирования подсистем и их частей.

1. Проектная оценка надежности системы

2.1 Документ содержит разделы:

* введение;
* исходные данные;
* методика расчета;
* расчет показателей надежности;
* анализ результатов расчета.
  1. В разделе «Введение» указывают:
* назначение расчета надежности системы;
* перечень оцениваемых показателей надежности;
* состав учитываемых при расчете факторов, а также принятые допущения и ограничения.
  1. В разделе «Исходные данные» приводят:
* данные о надежности (паспортные и справочные) элементов АС, учитываемые при расчете надежности системы;
* данные о режимах и условиях функционирования элементов АС;
* сведения об организационных формах, режимах, и параметрах эксплуатации АС.
  1. В разделе «Методика расчета» указывают обоснование выбора методики расчета и нормативно-технический документ, согласно которого проводят расчет, или краткое описание методики расчета и ссылку на источники, где она опубликована.
  2. В разделе «Расчет показателей надежности» указывают:
* Надёжностые структуры компонентов АС (комплекса технических средств, программного обеспечения и персонала) по всем оцениваемым функциям (функциональным подсистемам) АС;
* необходимые вычисления;
* результаты расчета.
  1. В разделе «Анализ результатов расчета» указывают:
* итоговые данные расчета по каждой оцениваемой функции (функциональной подсистеме) АС и каждому нормируемому показателю надежности;
* выводы о достаточности или недостаточности полученного уровня надежности АС по каждой оцениваемой функции (функциональной подсистеме) АС и, при необходимости, рекомендации по повышению надежности.
* Если в обоснованных случаях при оценке надежности АС нельзя учесть уровень надежности программного обеспечения АС и уровень надежности действий персонала АС, то в документе «Проектная оценка надежности системы» указывают сведения по оценке надежности АС только с учетом надежности комплекса технических средств, в том числе нестандартных.

1. Составление технического задания на программирование

Этап анализа и спецификации программ, жизненного цикла программных продуктов. При составлении технического задания требуется:

* определить платформу разрабатываемой программы – тип операционной системы;
* определить необходимость разработки программы, которую можно переносить на различные платформы;
* выбрать методы решения задачи;
* разработать обобщенный алгоритм решения комплекса задач, функциональную структуру алгоритма или состав объектов;
* определить требования к комплексу технических средств системы обработки информации, интерфейсу конечного пользователя;

1. Рабочая документация (рабочий проект)

На данном этапе осуществляется адаптация базовых средств программного обеспечения (операционной системы, СУБД, инструментальных сред конечного пользователя — текстовых редакторов, электронных таблиц и т.п.). Выполняется разработка программных модулей или методов обработки объектов – собственно программирование или создание программного кода. Проводятся автономная и комплексная отладка программного продукта, испытание работоспособности программных модулей и базовых программных средств. Для комплексной отладки готовится контрольный пример, который позволяет проверить соответствие возможностей программного продукта заданным спецификациям.

Основной результат работ этого этапа – создание эксплуатационной документации на программный продукт, включающей:

* описание применения – дает общую характеристику программного изделия с указанием сферы его применения, требований к базовому программному обеспечению, комплексу технических средств;
* руководство пользователя – включает детальное описание функциональных возможностей и технологии работы с программным продуктом. Данный вид документации ориентирован на конечного пользователя и содержит необходимую информацию для самостоятельного освоения и нормальной работы пользователя (с учетом требуемой квалификации пользователя);
* руководство программиста (оператора) – указывает особенности установки (инсталляции) программного продукта и его внутренней структуры – состав и назначение модулей, правила эксплуатации и обеспечения надежной и качественной работы программного продукта.

1. Структура программного обеспечения

В разделе "Структура программного обеспечения" приводят перечень частей программного обеспечения с указанием их взаимосвязей и обоснованием выделения каждой из них.  
Пример содержания:  
Наполнение этого раздела можно взять в ТЗ, пункте "4.1.1.1 Перечень подсистем, их назначение и основные характеристики".

1. Функции частей программного обеспечения

В разделе "Функции частей программного обеспечения" приводят назначение и описание основных функций для каждой части программного обеспечения.  
Пример содержания:  
Наполнение этого раздела можно взять в ТЗ, пункте "4.2 Требования к функциям (задачам), выполняемым системой".

1. Методы и средства разработки программного обеспечения

В разделе "Методы и средства разработки программного обеспечения" приводят перечень методов программирования и средств разработки программного обеспечения АС с указанием частей программного обеспечения, при разработке которых следует использовать соответствующие методы и средства.

Пример содержания:  
Проектирование структур баз данных АС «Кадры» выполнялось с использованием программного средства Microsoft Visio 2003 Enterprise Architect.

1. Операционная система

В разделе "Операционная система" указывают:

* Наименование, обозначение и краткая характеристика выбранной операционной системы и ее версии, в рамках которой будут выполнять разрабатываемые программы, с обоснованием выбора и указанием источников, где дано подробное описание выбранной версии;
* Наименование руководства, в соответствии с которым должна осуществляться генерация выбранного варианта операционной системы;
* Требования к варианту генерации выбранной версии операционной системы;

1. Средства, расширяющие возможности операционной системы

Наименование, обозначение и краткая характеристика средства с обоснованием необходимости его применения и указанием источника, где дано подробное описание выбранного средства.

Наименование руководства, в соответствии с которым следует настраивать используемое средство на конкретное применение.

Требования к настройке используемого средства.

1. Технологическая инструкция

Документ «Технологическая инструкция» разрабатывают на операцию или комплекс операций технологического процесса обработки данных.

В документе указывают наименование технологической операции (операций), на которую разработан документ, и приводят сведения о порядке и правилах выполнения операций (операции) технологического процесса обработки данных. В инструкции приводят перечень должностей персонала, на которые распространяется данная инструкция.

Номенклатуру технологических инструкций определяют, исходя из принятого процесса обработки данных. Структуру документа устанавливает разработчик в зависимости от содержания.

1. Руководство пользователя
2. Документ содержит разделы:

* введение;
* назначение и условия применения;
* подготовка к работе;
* описание операций;
* аварийные ситуации;
* рекомендации по освоению.

1. В разделе «Введение» указывают:

* область применения;
* краткое описание возможностей;
* уровень подготовки пользователя;
* перечень эксплуатационной документации, с которыми необходимо ознакомиться пользователю.

1. В разделе «Назначение и условия применения» указывают:

* виды деятельности, функции, для автоматизации которых предназначено данное средство автоматизации;
* условия, при соблюдении (выполнении, наступлении) которых обеспечивается применение средства автоматизации в соответствии с назначением (например, вид ЭВМ и конфигурация технических средств, операционная среда и общесистемные программные средства, входная информация, носители данных, база данных, требования к подготовке специалистов и т.п.).

1. В разделе «Подготовка к работе» указывают:

* состав и содержание дистрибутивного носителя данных;
* порядок загрузки данных и программ;
* порядок проверки работоспособности.

1. В разделе «Описание операций» указывают:

* описание всех выполняемых функций, задач, комплексов задач, процедур;
* описание операций технологического процесса обработки данных, необходимых для выполнения функций, комплексов задач (задач), процедур.

1. Для каждой операции обработки данных указывают:

* наименование;
* условия, при соблюдении которых возможно выполнение операции;
* подготовительные действия;
* основные действия в требуемой последовательности;
* заключительные действия;
* ресурсы, расходуемые на операцию.
* В описании действий допускаются ссылки на файлы подсказок, размещенные на магнитных носителях.

1. В разделе «Аварийные ситуации» указывают:

* действия в случае несоблюдения условий выполнения технологического процесса, в том числе при длительных отказах технических средств;
* действия по восстановлению программ и/или данных при отказе магнитных носителей или обнаружении ошибок в данных;
* действия в случаях обнаружения несанкционированного вмешательства в данные;
* действия в других аварийных ситуациях.

1. В разделе «Рекомендации по освоению» указывают рекомендации по освоению и эксплуатации, включая описание контрольного примера, правила его запуска и выполнения.

**13.03.2020**

**Создание модулей. Выбор метода разработки модуля**

Современные тенденции в разработке информационных систем требуют от проектировщиков закладывать в архитектуру систем возможность динамического расширения их функционала. И не смотря на существование огромного количества наработок в этом направлении, единого решения структуры модульного приложения нет. Использование же готового решения не всегда возможно в силу специфики языка программирования или разрабатываемой системы. Так же, готовые решения модульных систем не всегда доступны для изучения, а иногда излишне сложны.

Модули в разных системах, зачастую, имеют разные границы функциональности. В системе могут быть выделены некоторые, строго определенные точки расширения – некоторый функционал, дополняемый сторонними разработчиками. Или система может представлять собой лишь механизм управления модулями, а весь ее функционал реализуется отдельными модулями.

***Этапы проектирования модульных приложений***

Для разработки модульного приложения, прежде всего, необходимо выделить тот функционал, который должен расширяться с помощью модулей.

Далее разрабатываются интерфейсы, с помощью которых система будет обращаться к сторонним реализациям за этим функционалом.

Самым тонким моментом становится вопрос о том, как динамически добавлять реализации интерфейсов.

***Reflection***

В .Net Framework присутствует мощная технология reflection. Reflection позволяет программе отслеживать и модифицировать собственную структуру и поведение во время выполнения.

Применительно к нашей задаче, reflection позволяет загружать библиотеки в память и проверять все реализованные в ней классы на предмет реализации необходимых интерфейсов.

Т.е., можно добавлять библиотеки (с реализацией выделенных интерфейсов), представляющие собой модули, в специальную директорию и с помощью технологии reflection находить и инстанцировать необходимые классы.Это наиболее популярное решение, которое часто встречается на просторах интернета. Но у подобного подхода есть существенные недостатки, связанные с высокой ресурсоемкостью.

Загрузка сборки в память и перебор всех доступных в ней классов, в поисках реализаций интерфейсов, требует большого количества оперативной памяти, и ощутимого времени на выполнение перебора. Все усложняется в том случае, если реализация метода выходит за рамки одной библиотеки и имеет зависимости от сторонних библиотек. Тогда под перебор попадают сборки, вообще не содержащие реализаций необходимых интерфейсов и процессорное время, затрачиваемое на их исследование, тратится впустую.

***Структура модуля***

Очевидно, чтобы исключить лишние библиотеки из перебора, необходима дополнительная информация о модуле. Подобным источником информации может выступить текстовый файл, сопровождающий библиотеки модуля и предоставляющий информацию о них. Таким образом, перед нами встает задача разработки требований к структуре модуля, одним из пунктов которого можно предложить требование наличия файла, указывающего на главную библиотеку с реализацией интерфейсов и содержащего перечень всех необходимых зависимостей.

Простейшим решением устройства модуля может быть следующее:

1. Модуль представляет собой архив всех необходимых библиотек. В качестве алгоритма сжатия может выступать zip. Причем, как такового сжатия не требуется (бинарные библиотеки плохо поддаются архивированию), необходимо просто объединить все составляющие модуля в один файл.

2. Кроме библиотек, модуль должен содержать текстовый файл (называемый дескриптором), содержащий информацию о главной библиотеке, зависимостях, и, для повышения быстродействия путем исключения перебора, название реализуемого интерфейса вместе с полным именем класса его реализующего.

Для реализации дескриптора плагина представляется удобным воспользоваться XML.Добавление и удаление модулей

Добавление нового модуля в систему может происходить в следующей последовательности:

1. Системе передается полный путь файла с добавляемым модулем.

2. Добавляемый модуль проверяется на соответствие своему дескриптору: проверяется наличие всех указанных библиотек, наличие главного класса и реализация им указанного интерфейса.

3. В директории системы, отведенной под хранение модулей, создается новая поддиректория для добавляемого модуля. Все библиотеки модуля копируются в эту директорию.

4. Вычисляется уникальный идентификатор\* модуля (как вариант, можно взять хеш от имени и версии модуля).

5. Вся информация из дескриптора модуля и вычисленный идентификатор записываются в системный реестр модулей сохранения информации об использовании модуля в прошлой сессии работы в системе-

Рисунок 2 – Пример структуры xml-файла

**23.03.2020**

**Программирование модуля. Логическая проверка модуля**

Модуль — это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции) и, возможно, некоторые исполняемые операторы. Модули представляют собой прекрасный инструмент для разработки библиотек прикладных программ и мощное средство модульного программирования.

При разработке программного модуля целесообразно придерживаться следующего порядка:

* изучение и проверка спецификации модуля, выбор языка программирования;
* выбор алгоритма и структуры данных;
* программирование (кодирование) модуля;
* шлифовка текста модуля;
* проверка модуля;
* компиляция модуля.

Первый шаг программирования модуля в значительной степени представляет собой смежный контроль структуры программы снизу: изучая спецификацию модуля, разработчик должен убедиться, что она ему понятна и достаточна для разработки этого модуля. В завершении этого шага выбирается язык программирования: хотя язык программирования может быть уже предопределен для всего программного модуля, все же в ряде случаев (если система программирования это допускает) может быть выбран другой язык, более подходящий для реализации данного модуля (например, язык ассемблера).

На втором шаге разработки программного модуля необходимо выяснить, не известны ли уже какие-либо алгоритмы для решения поставленной и или близкой к ней задачи. И если найдется подходящий алгоритм, то целесообразно им воспользоваться. Выбор подходящих структур данных, которые будут использоваться при выполнении модулем своих функций, в значительной степени предопределяет логику и качественные показатели разрабатываемого модуля, поэтому его следует рассматривать как весьма ответственное решение.

На третьем шаге осуществляется построение текста модуля на выбранном языке программирования. Обилие всевозможных деталей, которые должны быть учтены при реализации функций, указанных в спецификации модуля, легко могут привести к созданию весьма запутанного текста, содержащего массу ошибок и неточностей. Искать ошибки в таком модуле и вносить в него требуемые изменения может оказаться весьма трудоемкой задачей. Поэтому весьма важно для построения текста модуля пользоваться технологически обоснованной и практически проверенной дисциплиной программирования. Впервые на это обратил внимание Дейкстра, сформулировав и обосновав основные принципы структурного программирования. На этих принципах базируются многие дисциплины программирования, широко применяемые на практике.

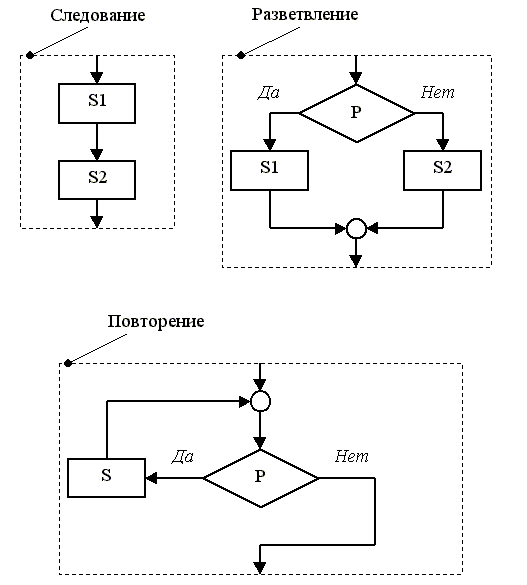


Рисунок 3 - Виды логики модулей

Следующий шаг разработки модуля связан с приведением текста модуля к завершенному виду в соответствии со спецификацией качества программного модуля. При программировании модуля разработчик основное внимание уделяет правильности реализации функций модуля, оставляя недоработанными комментарии и допуская некоторые нарушения требований к стилю программы. При шлифовке текста модуля он должен отредактировать имеющиеся в тексте комментарии и, возможно, включить в него дополнительные комментарии с целью обеспечить требуемые примитивы качества. С этой же целью производится редактирование текста программы для выполнения стилистических требований.

Шаг проверки модуля представляет собой ручную проверку внутренней логики модуля до начала его отладки (использующей выполнение его на компьютере), реализует общий принцип, сформулированный для обсуждаемой технологии программирования, о необходимости контроля принимаемых решений на каждом этапе разработки программного модуля.

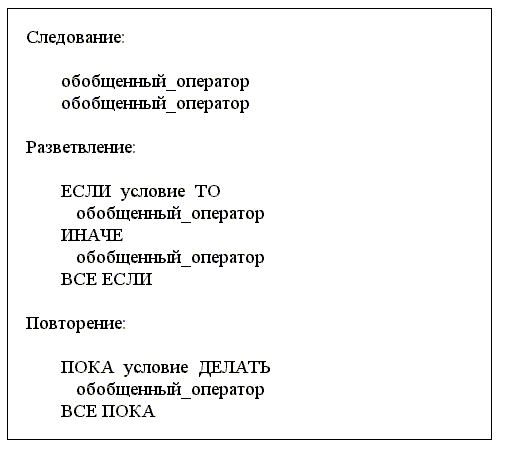


Рисунок 3 - Внешнее оформление описания процедуры или функции

И, наконец, последний шаг – завершение проверки модуля (с помощью компилятора) и переход к процессу отладки модуля.

**24.03.2020**

**Компиляция модуля. Отладка и тестирование модулей**

Компиля́ция — сборка программы, включающая [трансляцию](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) всех модулей программы, написанных на одном или нескольких исходных [языках программирования высокого уровня](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) и/или [языке ассемблера](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0), в эквивалентные программные модули на [низкоуровневом языаке](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), близком [машинному коду](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4).

***Виды компиляции:***

* *Пакетная*.
* *Построчная*.
* *Условная*.

**25.03.2020**

**Отладка модуля с целью выявления логических ошибок**

При разработке программ наиболее трудоемким является этап отладки и тестирования программ. Цель тестирования, т.е. испытания программы, заключается в выявлении имеющихся в программе ошибок. Цель отладки состоит в выявлении и устранении причин ошибок.

Отладку программы начинают с составления плана тестирования. Такой план должен представлять себе любой программист. Составление плана опирается на понятие об источниках и характере ошибок. Основными источниками ошибок являются недостаточно глубокая проработка математической модели или алгоритма решения задачи; нарушение соответствия между схемой алгоритма или записью его на алгоритмическом языке и программой, записанной на языке программирования; неверное представление исходных данных на программном бланке; невнимательность при наборе программы и исходных данных на клавиатуре устройства ввода.

Нарушение соответствия между детально разработанной записью алгоритма в процессе кодирования программы относится к ошибкам, проходящим вследствие невнимательности программиста. Отключение внимания приводит и ко всем остальным ошибкам, возникающим в процессе подготовки исходных данных и ввода программы в ЭВМ. Ошибки, возникающие вследствие невнимательности, могут иметь непредсказуемые последствия, так как наряду с потерей меток и описаний массивов, дублированием меток, нарушением баланса скобок возможны и такие ошибки, как потеря операторов, замена букв в обозначениях переменных, отсутствие определений начальных значений переменных, нарушение адресации в массивах, сдвиг исходных данных относительно полей значений, определенных спецификациями формата.

В план тестирования обычно входят следующие этапы:

1. Сравнение программы со схемой алгоритма.
2. Визуальный контроль программы на экране дисплея или визуальное изучение распечатки программы и сравнение ее с оригиналом на программном бланке.
3. Трансляция программы на машинных язык.
4. Редактирование внешних связей и компоновка программы.
5. Выполнение программы.
6. Тестирование программы.

**26.03.2020**

**Верификация и аттестация модуля. Разработка системы тестов.**

Верификацией и аттестацией называют процессы проверки и анализа, в ходе которых проверяется соответствие программного обеспечения своей спецификации и требованиям заказчиков. Верификация и аттестация охватывают полный жизненный цикл ПО – они начинаются на этапе анализа требований и завершаются проверкой программного кода на этапе тестирования готовой программной системы.

Верификация и аттестация не одно и то же, хотя их легко перепутать. Кратко различие между ними можно определить следующим образом:

• верификация отвечает на вопрос, правильно ли создана система;

• аттестация отвечает на вопрос, правильно ли работает система.

Согласно этим определениям, верификация проверяет соответствие ПО системной спецификации, в частности функциональным и нефункциональным требованиям. Аттестация– более общий процесс. Во время аттестации необходимо убедиться, что программный продукт соответствует ожиданиям заказчика. Аттестация проводится после верификации, для того чтобы определить, насколько система соответствует не только спецификации, но и ожиданиям заказчика.

Как уже отмечалось ранее, на ранних этапах разработки ПО очень важна аттестация системных требований. В требованиях часто встречаются ошибки и упущения; в таких случаях конечный продукт, вероятно, не будет соответствовать ожиданиям заказчика. Но, конечно, аттестация требований не может выявить все проблемы в спецификации требований. Иногда недоработки и ошибки в требованиях обнаруживаются только после завершения реализации системы.

В процессах верификации и аттестации используются две основные методики проверки и анализа систем.

1. Инспектирование ПО. Анализ и проверка различных представлений системы, например документации спецификации требований, архитектурных схем или исходного кода программ. Инспектирование выполняется на всех этапах процесса разработки программной системы. Параллельно с инспектированием может выполняться автоматический анализ исходного кода программ и соответствующих документов. Инспектирование и автоматический анализ – это статические методы верификации и аттестации, поскольку им не требуется исполняемая система.

2. Тестирование ПО. Запуск исполняемого кода с тестовыми данными и исследование выходных данных и рабочих характеристик программного продукта для проверки правильности работы системы. Тестирование – это динамический метод верификации и аттестации, так как применяется к исполняемой системе.

На рис. 4 показано место инспектирования и тестирования в процессе разработки ПО. Стрелки указывают на те этапы процесса разработки, на которых можно применять данные методы. Согласно этой схеме, инспектирование можно выполнять на всех этапах процесса разработки системы, а тестирование – в тех случаях, когда создан прототип или исполняемая программа.

К методам инспектирования относятся: инспектирование программ, автоматический анализ исходного кода и формальная верификация. Но статические методы могут проверить только соответствие программ спецификации, с их помощью невозможно проверить правильность функционирования системы. Кроме того, статическими методами нельзя проверить такие нефункциональные характеристики, как производительность и надежность. Поэтому для оценивания нефункциональных характеристик проводится тестирование системы.

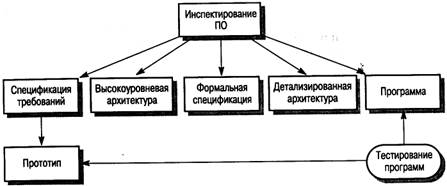


Рисунок 4 - Статическая и динамическая верификация и аттестация

Несмотря на широкое применение инспектирования ПО, преобладающим методом верификации и аттестации все еще остается тестирование. Тестирование – это проверка работы программ с данными, подобными реальным, которые будут обрабатываться в процессе эксплуатации системы. Наличие в программе дефектов и несоответствий требованиям обнаруживается путем исследования выходных данных и выявления среди них аномальных. Тестирование выполняется на этапе реализации системы (для проверки соответствия системы ожиданиям разработчиков) и после завершения ее реализации.

На разных этапах процесса разработки ПО применяют различные виды тестирования.

1. Тестирование дефектов проводится для обнаружения несоответствий между программой и ее спецификацией, которые обусловлены ошибками или дефектами в программах. Такие тесты разрабатываются для выявления ошибок в системе, а не для имитации ее работы.

2. Статистическое тестирование оценивает производительность и надежность программ, а также работу системы в различных режимах эксплуатации. Тесты разрабатываются так, чтобы имитировать реальную работу системы с реальными входными данными. Надежность функционирования системы оценивается по количеству сбоев, отмеченных в работе программ. Производительность оценивается по результатам измерения полного времени выполнения операций и времени отклика системы при обработке тестовых данных.

Главная цель верификации и аттестации – удостовериться в том, что система "соответствует своему назначению". Соответствие программной системы своему назначению отнюдь не предполагает, что в ней совершенно не должно быть ошибок. Скорее, система должна достаточно хорошо соответствовать тем целям, для которых планировалась. Уровень необходимой достоверности соответствия зависит от назначения системы, ожиданий пользователей и условий на рынке программных продуктов.

1. Назначение ПО. Уровень достоверности соответствия зависит от того, насколько критическим является разрабатываемое программное обеспечение по тем или иным критериям. Например, уровень достоверности для систем, критическим по обеспечению безопасности, должен быть значительно выше аналогичного уровня достоверности для опытных образцов программных систем, разрабатываемых для демонстрации некоторых новых идей.

2. Ожидания пользователей. Следует с грустью отметить, что в настоящее время у большинства пользователей невысокие требования к программному обеспечению. Пользователи настолько привыкли к отказам, происходящим во время работы программ, что не удивляются этому. Они согласны терпеть сбои в работе системы, если преимущества ее использования компенсируют недостатки. Вместе с тем с начала 1990-х годов терпимость пользователей к отказам в работе программных систем постепенно снижается. В последнее время создание ненадежных систем стало практически неприемлемым, поэтому компаниям, занимающимся разработкой программных продуктов, необходимо все больше внимания уделять верификации и аттестации программного обеспечения.

3. Условия рынка программных продуктов. При оценке программной системы продавец должен знать конкурирующие системы, цену, которую покупатель согласен заплатить за систему, и назначенный срок выхода этой системы на рынок. Если у компании-разработчика несколько конкурентов, необходимо определить дату выхода системы на рынок до окончания полного тестирования и отладки, иначе первыми на рынке могут оказаться конкуренты. Если покупатели не желают приобретать ПО по высокой цене, возможно, они согласны терпеть большее количество отказов в работе системы. При определении расходов на процесс верификации и аттестации необходимо учитывать все эти факторы.

Как правило, в ходе верификации и аттестации в системе обнаруживаются ошибки. Для исправления ошибок в систему вносятся изменения. Этот процесс отладки обычно интегрирован с другими процессами верификации и аттестации. Вместе с тем тестирование (или более обобщенно – верификация и аттестация) и отладка являются разными процессами, которые имеют различные цели.

1. Верификация и аттестация – процесс обнаружения дефектов в программной системе.

2. Отладка – процесс локализации дефектов (ошибок) и их исправления (рис. 5).

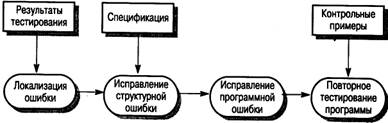


Рисунок 5 - Процесс отладки

Простых методов отладки программ не существует. Опытные отладчики обнаруживают ошибки путем сравнения шаблонов тестовых выходных данных с выходными данными тестируемых систем. Чтобы определить местоположение ошибки, необходимы знания о типах ошибок, шаблонах выходных данных, языке программирования и процессе программирования. Очень важны знания о процессе разработке ПО. Отладчикам известны наиболее распространенные ошибки программистов (например, связанные с пошаговым увеличением значения счетчика). Также учитываются ошибки, типичные для определенных языков программирования, например связанные с использованием указателей в языке С.

Определение местонахождения ошибок в программном коде не всегда простой процесс, поскольку ошибка необязательно находится возле того места в коде программы, где произошел сбой. Чтобы локализовать ошибки, программист-отладчик разрабатывает дополнительные программные тесты, которые помогают выявить источник ошибки в программе. Может возникнуть необходимость в ручной трассировке выполнения программы.

Интерактивные средства отладки являются частью набора средств поддержки языка, интегрированных с системой компиляции программного кода. Они обеспечивают специальную среду выполнения программ, посредством которой можно получить доступ к таблице идентификаторов, а оттуда к значениям переменных. Пользователи часто контролируют выполнение программы пошаговым способом, последовательно переходя от оператора к оператору. После выполнения каждого оператора проверяются значения переменных и выявляются возможные ошибки.

Обнаруженная в программе ошибка исправляется, после чего необходимо снова проверить программу. Для этого можно еще раз выполнить инспектирование программы или повторить предыдущее тестирование. Повторное тестирование используется для того, чтобы убедиться, что сделанные в программе изменения не внесли в систему новых ошибок, поскольку на практике высокий процент "исправления ошибок" либо не завершается полностью, либо вносит новые ошибки в программу.

В принципе во время повторного тестирования после каждого исправления необходимо еще раз запускать все тесты, однако на практике такой подход оказывается слишком дорогостоящим. Поэтому при планировании процесса тестирования определяются зависимости между частями системы и назначаются тесты для каждой части. Тогда можно трассировать программные элементы с помощью специальных контрольных примеров (контрольных данных), подобранных для этих элементов. Если результаты трассировки задокументированы, то для проверки измененного программного элемента и зависимых от него компонентов можно использовать только некоторое подмножество всего множества тестовых данных.

***Планирование верификации и аттестации***

Верификация и аттестация – дорогостоящий процесс. Для больших систем, например систем реального времени со сложными нефункциональными ограничениями, половина бюджета, выделенного на разработку системы, тратится на процесс верификации и аттестации. Поэтому очевидна необходимость тщательного планирования данного процесса.

Планирование верификации и аттестации, как один из этапов разработки программных систем, должно начинаться как можно раньше. На рис. 3 показана модель разработки ПО, учитывающая процесс планирования испытаний. Здесь планирование начинается еще на этапах создания спецификации и проектирования системы. Данную модель иногда называют V-моделью (чтобы увидеть букву V, необходимо повернуть рис. 3 на 90°). На этой схеме также показано разделение процесса верификации и аттестации на несколько этапов, причем на каждом этапе выполняются соответствующие тесты.

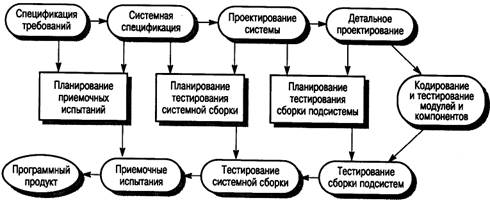


Рисунок 6 - Планирование испытаний в процессе разработки и тестирования

В процессе планирования верификации и аттестации необходимо определить соотношение между статическими и динамическими методами проверки системы, определить стандарты и процедуры инспектирования и тестирования ПО, утвердить технологическую карту проверок программ (см. раздел 19.2) и составить план тестирования программ. Чему уделить больше внимания – инспектированию или тестированию, зависит от типа разрабатываемой системы и опыта организации. Чем более критична система, тем больше внимания необходимо уделить статическим методам верификации.

В плане верификации и аттестации основное внимание уделяется стандартам процесса тестирования, а не описанию конкретных тестов. Этот план предназначен не только для руководства, он в основном предназначен специалистам, занимающимся разработкой и тестированием систем. План дает возможность техническому персоналу получить полную картину испытаний системы и в этом контексте спланировать свою работу. Кроме того, план предоставляет информацию менеджерам, отвечающим за то, чтобы у группы тестирования были все необходимые аппаратные и программные средства.

План испытаний не является неизменным документом. Его следует регулярно пересматривать, так как тестирование зависит от процесса реализации системы. Например, если реализация какой-либо части системы не завершена, то невозможно провести тестирование сборки системы. Поэтому план необходимо периодически пересматривать, чтобы сотрудников, занятых тестированием, можно было использовать на других работах.

**24.01.2020**

**Выбор критерия завершенности тестирования. Апробация работы модуля**

При проведении тестирования встает вопрос о том, когда завершить тестирование, когда разрабатываемое ПС достигло того уровня надежности, которое может удовлетворить будущих пользователей.

В основном на практике придерживаются следующих двух критериев: когда время, отведенное по графику на тестирование, истекло; когда все тесты выполняются без выявления ошибок.

Оба этих критерия недостаточно точны и логичны, так как первому можно удовлетворить, ничего не делая, а второй зависит от качества тестового набора данных.

Иногда используют критерий, основанный в значительной степени на здравом смысле и информации о количестве ошибок, полученных в процессе тестирования. Для этого строят график зависимости количества ошибок и времени их появления. По форме полученной кривой можно определить, стоит продолжать тестирование или нет.

На рисунке 7 приведены 2 графика, из которых видно, что если с увеличением времени тестирования число ошибок растет (левый), то тестирование необходимо про­должать. Если в процессе тестирования в определенный момент наступило снижение числа выявленных ошибок, постепенно стремится к нулю или достигло нуля, то понятно, что процесс тестирования можно завершать (правый).

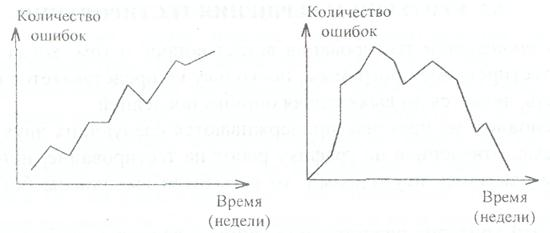


Рисунок 7 – Отношение времени тестирования к количеству ошибок

*На практике могут быть использованы следующие критерии:*

1. Время, отведенное по графика на тестирование, истекло.

2. Когда все тесты выполняются без выявления ошибок.

3. На основании графика, в зависимости количества ошибок и времени их появления.

4. Если с увеличением времени тестирования, количество ошибок растет, то тестирование необходимо продолжать. Если в определенный момент наступило снижение числа выделенных ошибок и постепенно достигло малого значения, то процесс тестирования можно завершить.

5. Основан на определение критерия завершения тестирования по количественным показателям надежности, которые рассчитываются по моделям надежности.

*Модель надежности программного обеспечения (рисунок 8)* – математическая модель, построенная для оценки надежности ПО от некоторых определенных параметров.

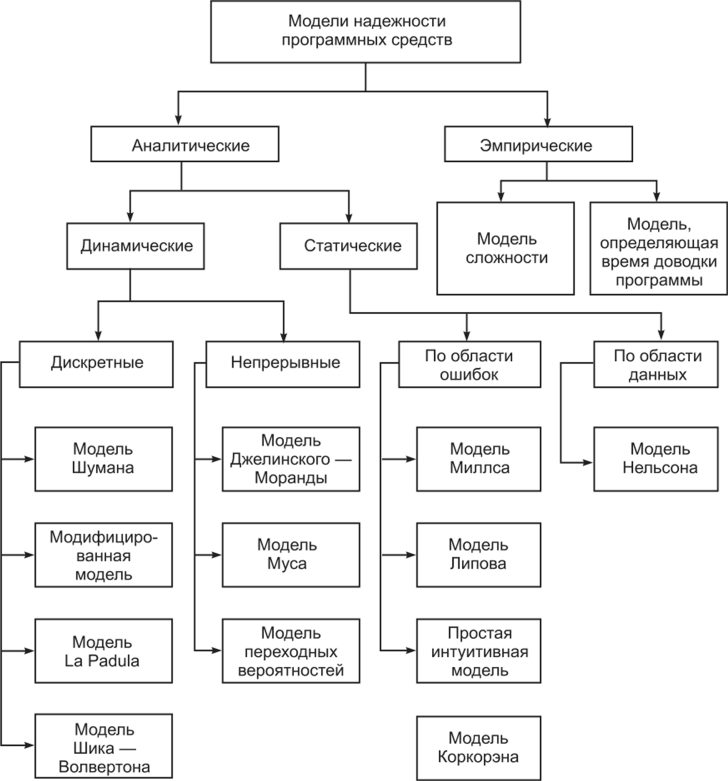
Аналитические модели дают возможность рассчитать, основываясь на данных в процессе тестирования. Аналитические модели представлены двумя группами: динамические и статические модели.

Рисунок 8 – Модели надёжности программных средств

***Динамические модели:***

Поведение программного средства оценивается через появление отказов.

***Статические модели:***

Учитывают количество ошибок от количества тестовых прогонов. По области ошибок. Зависимость количества ошибок от характеристики входных данных по области.

Для использования динамических моделей. Если фиксируются интервалы каждого отказа, то получается непрерывная картина появления отказа во времени.

***Аналитические модели:***

1. Определение предположений, связанных с процедурой тестирования программного средства.

2. Разработка и выбор аналитической модели, базирующийся на предположениях о процедурах тестирования.

3. Выбор параметра модели с использованием полученных данных.

4. Применение модели. Расчет количественных показателей по модели.

***Модель Шумана:***

Исходные данные для модели Шумана собираются в процессе тестирования программного средства, в течение фиксированных или случайных временных интервалов. Каждый интервал — это стадия, на которой выполняется последовательность тестов и фиксируется число ошибок. Тестирование проводится в несколько этапов. Каждый этап представляет собой выполнение программы на полном комплексе разработанных тестовых данных. Выявленные ошибки регистрируются, но не исправляются. По завершении этапа на основе на очередном этапе тестирования может быть использована модель Шумана для расчета количественных показателей надежности и только после этого исправляются ошибки, обнаруженные на предыдущем этапе. При использовании модели Шумана предполагается, что исходное количество ошибок в программе постоянно и в процессе тестирования может уменьшаться по мере того, как ошибки выявляются и исправляются. Новые ошибки при корректировке не вносятся. Скорость обнаружения пропорционально числу ошибок. Общее число машинных инструкций в рамках одного этапа тестирования постоянно.

***Понятие апробации:***

Необходимый шаг созревания работы — ее апробация. Верная апробация исследования — один из критериев его корректности, состоятельности, истинности итогов, один из самых реальных методов не допустить серьёзных ошибок, перекосов, преодолеть индивидуальные пристрастия исследователей, своевременно подкорректировать и поправить допущенные ошибки и недочеты. Слово «апробация» латинского происхождения и означает «одобрение, утверждение, установление качеств» (не путать с опробованием, т. е. проверкой на практике).

***Формы апробации:***

Апробация может проходить в форме прилюдных отчетов, дискуссий, дискуссий, и еще в форме рецензирования (устного либо письменного) представленных работ. Официальная апробация исполненных работ нередко связана с их общественной охраной (защита плана, отчета, курсовой или же выпускной работы, диссертации). Присутствует и играет не самую последнюю роль и неофициальная апробация: разговоры, споры с экспертами и коллегами. Само собой, для апробации обязан быть представлен хотя бы в первоначальном варианте оформленный текст доклада, отчета, известия, плана, диссертации, но о процедуре литературного дизайна речь пойдет чуток ниже. Апробацию диссертации надо начинать сразу после начала работы над ней, в данном случае, кандидат сумеет получить беспристрастную оценку каждого шага проделанного им исследования, произведенных выводов и практических советов, что в нем содержатся. Проинформировать научную общественность о материалах собственной диссертации и работе над ней кандидат может, принимая участие в разных конференциях и симпозиумах, выступая на заседаниях кафедры, участвуя в разных событиях, проводимых научным обществом, кроме того у него есть возможность готовить и подталкивать во всевозможные органы собственные предложения по исследуемой теме. Помимо того, кандидат может публиковать книги, тезисы отчетов на конференциях, депонированные части собственных научных изысканий. Работу надо каждый день оговаривать с сотрудниками, преподавательским составом и научными работниками того учебного заведения, в каком готовится диссертация.

***Значение апробации:***

Конечно, апробация включает осмысление и учет образующихся вопросов, положительных и неблагоприятных оценок, отрицаний и советов. Она провоцирует доработку, более глубочайшее и аргументированное объяснение либо пересмотр ряда положений исследования и приемов подтверждения, может помочь или утвердиться в признании истинности защищаемых положений, или подкорректировать либо пересмотреть их. Мы ведем речь, по существу, об итоговой апробации исследования, хотя в общем-то может быть полезна, часто просто нужна поэтапная апробация начальных положений, гипотез, методики исследования, его переходных результатов. Положительные отзывы, в случае если они не носят комплиментарного характера, приносят удовлетворение, дают уверенность, могут помочь раскрыть возможности последующих поисков. Менее приятны, но ничуть не ниже, могут быть полезны критические замечания, рекомендации и предложения. Чрезвычайно полезным может быть глубокий анализ образовавшихся вопросов.

Есть основания признать, что могут быть полезны в конечном счете всевозможные образовавшиеся в ходе представления работы вопросы. Все связано с тем, как они воспринимаются и применяются. Из числа вопросов можно выделить уточняющие, связанные с недопониманием, неполным или же неверным осознанием изложенного. Они побуждают к уточнению, поиску более точных формулировок, доработке стиля, т.е. могут помочь сделать изложение более точным и убедительным. Вопросы, дополняющие предполагают запрос на вспомогательную информацию об источниках и перспективах становления, прецедентах, первопричинах, следствиях и т.д. Они побуждают исследователя использовать новейшие прецеденты, отдать вспомогательные оценки и мониторинги. Такие вопросы расширяют спектр оценок и раскладов. Вопросы, корректирующие побуждают к уточнению, усиливают аргументацию, уничтожают двусмысленность. Проблемные вопросы вскрывают трудности, нацеливают на более основательную интерпретацию, выводят на свежие проблемы и задачи. Стоит посоветовать чрезвычайно тщательно и дружественно воспринимать все оценки, рекомендации, советы и соратников, и оппонентов, в том числе и критические. Впрочем, воплотить стоит те из них, что могут помочь углубить изыскание, увеличить его корректность и доказательность, что не противоречит принятой концепции и вовсе не устроят работу эклектичной. Стремление же угодить всем, отреагировать на каждое замечание может усугубить или повредить изложению результатов. Апробация итогов считается одним из главных разделов введения диссертации. Оцениваются не только последствия работы, но и методика исследования, которую применил исследователь и промежуточные итоги его работы. Благодаря апробации, кандидат может переосмыслить собственные научные исследования, глубоко их подвергнуть доработке, удостовериться в надобности пересмотра каких-либо их положений. Благодаря апробации, кандидат может не только быть полноценным научным работником, но и обучиться без ошибок готовить отчеты и концерты, вести научную обсуждение вопроса, благодаря чему, у него появится возможность как следует приготовиться к защите диссертации и уверенно провести ее. Главные результаты приобретенные в диссертации представлялись соискателем в период симпозиумов и научно-практических конференций, а по ее теме публиковались учебные пособия и заметки, в каких были отражены ключевые теоретические основы работы и ее последствия, кроме того вполне возможно сказать, что последствия диссертации были интегрированы в доклад о научно-исследовательской работе.

Результаты диссертаций, имеющих прикладной и частично – теоретический нрав могут применяться во множества секторах экономики этнического хозяйства, им предоставляется возможность употребляться еще до защиты самой диссертации. Это внедрение считается введением эффектов работы и обязано отражаться в ее введении. Введение обязано быть документально утверждено той организацией, которая применила в собственной работы приобретенные соискателем последствия и возымела от их применения какой-то эффект. Эффективность внедрения результатов исследования зависит от того, как в диссертации разработаны теоретические и методические положения, а еще тем, доведены ли они до точных советов, представленных в форме нормативов, руководств и методик. Данные советы могут касаться улучшения структуры производства, нормативов временных расходов, кроме того, это могут быть аннотации по использованию всевозможных программ. Введением может считаться и внедрение результатов исследования в учебном процессе, через включение их в разные учебные материалы и пособия. Введением именуется предоставление точных итогов исследования потребителю в комфортной для него форме, способной увеличить эффективность его работы, введение постоянно обязано быть без ошибок документально оформлено.

***Апробация работы модуля:***

Апробация первого модуля тестирования прошла в рамках учебного заведения «Первого Московского образовательного комплекса» в виде презентации материала.

**24.01.2020**

**Предметная область**

Сегодня на практике я занималась анализом предметной области.

Предметную область можно определить как сферу человеческой деятельности, выделенную и описанную согласно установленным критериям. В описываемое понятие должны входить сведения об ее элементах, явлениях, отношениях и процессах, отражающих различные аспекты этой деятельности. В описании предметной области должны присутствовать характеристики возможных воздействий окружающей среды на элементы и явления предметной области, а также обратные воздействия этих элементов и явлений на среду. Работа по изучению и анализу предметной области: проектировании интеллектуальных систем оказывает решающее влияние на эффективность ее работы.

Специфика предметной области может оказывать существенное влияние на характер функционирования проектируемой интеллектуальной системы, выбор метода представления знаний, способов рассуждения о знаниях, и т. д.

Предметную область можно определить как объект или производственную систему со всем комплексом понятий и знаний о ее функционировании. При исследовании проблемной области необходимы знания о задачах, решаемых в производственной системе, и стоящих перед ней целях. Определяются также возможные стратегии управления и эвристические знания, используемые в процессе эксплуатации производственной системы.

**27.01.2020**

**Анализ предметной области**

Одна из первых задач, с решением которых сталкивается разработчик программной системы — это изучение, осмысление и анализ предметной области. Дело в том, что предметная область сильно влияет на все аспекты проекта: требования к системе, взаимодействие с пользователем, модель хранения данных, реализацию и т.д.

Анализ предметной области, позволяет выделить ее сущности, определить первоначальные требования к функциональности и определить границы проекта (рис. 9). Модель предметной области должна быть документирована, храниться и поддерживаться в актуальном состоянии до этапа реализации. Для документирования могут быть использованы различные средства.



Рисунок 9 – Анализ предметной области

Для управления обсуждением области действия проекта можно использовать методику "будет - не будет". В простейшем случае — это список с двумя столбцами, в одном из которых записывается, что проект будет делать, а во втором - что не входит в проект. Такой список, формируется заинтересованными лицами при рассмотрении каждой бизнес-цели проекта, используя любую технику, например метод "мозгового штурма" (см. тему "Выявление требований"). Полученные характеристики позволяют четко определить границы проекта и довольно просто преобразуются в предположения, которые фиксируются в документе.

Функциональная область действия определяет услуги, предоставляемые системой, и вначале до конца неизвестны. В определении услуг системы может помочь список "Действующее лицо/Цель", в котором перечислены все цели пользователя, поддерживаемые системой. При его разработке в первую графу вписываются имена основных действующих лиц, т.е. тех, кто имеет цели, во вторую графу - цель каждого действующего лица, а в третью - приоритет или предположение о том, в какую версию войдет эта услуга. Формы списков приведены на рисунке.

Для определения основных функций продукта можно использовать, например, краткое описание варианта использования. Описание каждой функции можно представить также в виде списка, состоящего из трех граф: действующее лицо, цель и краткое описание варианта использования.

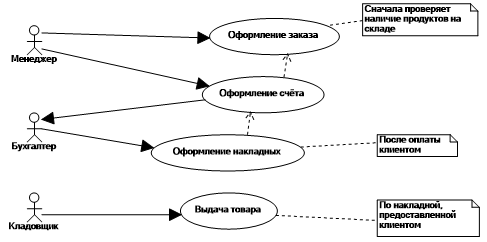


Рисунок 10 – Диаграмма вариантов использования

Анализ предметной области является основой для анализа осуществимости проекта и определения образа (концепции) продукта и границ проекта.

**28.01.2020**

**Обоснование выбора технических средств**

На основании поставленной задачи делается вывод о необходимости использования той или иной программы. После чего выбирается выбор технического средства (ПК).

1. Технические характеристики: -объем памяти; -быстродействие; -состав аппаратных средств;
2. Удобство и простота использования;
3. Стоимостные затраты;

При выборе комплекса технических средств необходимо учитывать следующие факторы:

1. стоимость приобретения технических средств;
2. стоимость использования технических средств;
3. надежность технических средств и срок службы;
4. удобство комплекса технических средств (КТС);
5. производительность и быстродействие КТС;

**29.01.2020**

**Построение и обоснование модели проекта**

Информационное обеспечение является базой, на которой строится вся управленческая деятельность. Информацию здесь следует рассматривать как некую совокупность различных сообщений, сведений, данных о соответствующих предметах, явлениях, процессах, отношениях и т.д. Эти сведения, будучи собранными, систематизированными и преобразованными в пригодную для использования форму играют в управлении исключительную роль. Необходимым условием для успешного функционирования любого промышленных предприятий является нормальная работа следующих процессов: - целенаправленный сбор, первичная обработка информации; - организация каналов доступа пользователей к собранной информации; - своевременное использование собранной информации для принятия решений. Основная проблема сбора необходимой информации состоит в том, чтобы обеспечить: - полноту, адекватность, непротиворечивость и целостность информации; - минимизацию технологического запаздывания между моментом зарождения информации и тем моментом, когда к информации может начаться доступ. Обеспечить это можно только современными автоматизированными методиками, базирующимися на основе информационных систем. Крайне важно, чтобы собранная информация была структурирована с учетом потребностей потенциальных пользователей и хранилась в форме, позволяющей использовать современные технологии доступа и обработки. Разработка информационных систем управления (ИСУ) является довольно сложным процессом, который требует значительного времени и ресурсов. Современные крупные проекты ИСУ характеризуются, как правило, следующими особенностями: - сложность описания (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов; - наличие совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (например, традиционных приложений, связанных с обработкой транзакций и решением регламентных задач, и приложений аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема); - отсутствие прямых аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем; - разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств; - существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ИСУ. Для успешной реализации проекта объект проектирования (ИСУ) должен быть прежде всего адекватно описан, должны быть построены полные и функциональные непротиворечивые и информационные модели ИСУ. Кроме того, в процессе создания и функционирования ИСУ информационные потребности пользователей могут изменяться или уточняться, что усложняет разработку и сопровождение таких систем. В настоящее время один из наиболее сложных и важных этапов разработки ИСУ, этап построение информационной модели, остается во многом не формализованным. Начальные фазы проекта имеют решающее влияние на достигаемый результат, так как в них принимаются основные решения, определяющие качество информационной системы. Доля вклада в конечный результат концептуальной фазы достигает 30%.

В результате проведения первичной обработки получаем не массу несистематизированной первичной информации, а информационную базу, содержащую данные о предприятии, пригодную для дальнейшего непосредственного автоматического анализа. Именно такое представление результатов обследования мы будем использовать в дальнейшей работе. Поставим задачу построения информационной модели, которая позволяла бы отражать не только связи между структурными подразделениями предприятия и их вес, но и оценивать суть происходящих в организации процессов. В данном случае – какие операции (функции) над информацией (документами) выполняются внутри организации. В дальнейшем будем называть такую информационную модель функциональной ориентированной. Получение такой модели позволит ставить задачу оптимизации организационной структуры предприятия по новым критериям, например, по загруженности отдельных функций, по равномерной загрузки и т.д. Полученная функционально-ориентированная модель позволит реализовать методику построения организационной модели предприятия «снизу»: на первом этапе выявить весь перечень функций, которые должны быть реализованы на данном предприятии, для эффективного управления и достижения поставленной цели; определить внутренние и внешние связи между функциями; оценить количество информации проходящей по этим связям; провести реорганизацию отделов и служб путем перераспределения данных функций по признаку подобия функций.

В настоящее время наибольшее распространение получили модели для следующих классов задач: - составление расписаний и календарное планирование; - массовое обслуживание; - распределение; - управление запасами; - износ и замена оборудования; - конфликтные ситуации. В поддержку создания служб и систем управления потоками работ предлагаются методологии, стандарты и специализированное программное обеспечение, образующие в комплексе инструментальные средства разработчика. Методология – это совокупность методов, применяемых в жизненном цикле разработки процесса и объединенных одним общим подходом. В настоящее время получила распространение методология SADT. (Structured Analysis and Design Tecchnique) ─ методология структурного анализа и проектирования, которые дают ряд преимуществ в системах управления: - формализацию описания потоков работ; - переносимость: модели процессов, созданные в рамках одной системы, могут работать под управлением другой системы; - универсальность: применение единого механизма описания управления потоками работ в различных сферах деятельности. На данный момент разработан ряд стандартов для описания реальных потоков работ, которые можно разбить на две категории: - Графовые модели, отражающие древовидную структуру процесса. - Блочные модели, наиболее приближенные к блочной структуре языков программирования. Итак, модели, указанные выше, в ряде случаев удобно использовать для эффективного описания окружения системы и ее функционирования. Между тем, по большому счету определяющей в любой системе является информация о потоках, обслуживающих систему. Поэтому имеет смысл исследовать не просто модель системы, а ее информационную модель (ИМ) детализированную до уровня функций и ее заполняемость информационными потоками, а не блоков системы. Такую модель можно называть функционально ориентированной ИМ (ФИМ). Использование ФИМ важно и для построения модели функциональной структуры системы. Использование ФИМ позволяет ставить и решать новые задачи на уровне организационной и функциональной структуры, например, определять загрузку функций, перераспределять (оптимизировать) документооборот между отдельными функциями с целью обеспечения их полной загрузки и т.д. Прежде, чем обсуждать эффективность ФИМ надо обратить внимание на то, что само исходное понятие информации оказывается до сих пор неоднозначно.

**30.01.2020**

**Проектирование и разработка интерфейса пользователя**

Что такое UI? Интерфейс — общая граница между двумя функциональными объектами, требования к которой определяются стандартом; совокупность средств, методов и правил взаимодействия (управления, контроля и т. д.) между элементами системы (источник: wikipedia.org). Это точное, но скучноватое определение. Пользовательский интерфейс (UI) — это «способ, которым вы выполняете какую-либо задачу с помощью какого-либо продукта, а именно совершаемые вами действия и то, что вы получаете в ответ». В повседневной жизни мы постоянно сталкиваемся с интерфейсами. Это и сайты соцсетей, и элементы управления в салоне автомобиля, и пульт ДУ для телевизора, и голосовое управление умным домом, и панель кнопок в лифте. Выходит, мы используем один продукт для управления другим продуктом. Но давайте не будем перечислять все явления в нашей жизни, а поговорим непосредственно о веб-сервисах и приложениях и о том, как сделать их использование удобным. Мы разобрались, что такое пользовательский интерфейс, но пока неясно, насколько он нам нужен и стоит ли тратить деньги на его разработку и дизайн. Поэтому на минуту вернёмся к лифтам. Лифт удобен. Он позволяет добраться до верхних этажей с минимумом временных и физических затрат. Но какой от него был бы толк, если бы мы не могли легко его открыть, выбрать этаж и при необходимости остановиться раньше времени? Всё перечисленное осуществимо, но каким путём? Позволит ли рычажное управление сделать пассажиру необходимые действия или разумнее будет использовать кнопочное управление? Тут и появляется важный момент: какой интерфейс управления лифтом выбрать. Современные веб-сервисы и системы в корне меняют работу людей. Существуют, например, системы управления театральными механизмами, которые упрощают работу с электронным оборудованием, лишая необходимости постоянно находиться у аппаратуры. Но едва ли работа упростится, если у таких систем будет слишком сложный интерфейс. Получается, что программный интерфейс не только решает нашу проблему взаимодействия с приложением, но и делает это взаимодействие максимально комфортным. Нам важно наличие интерфейса, позволяющего при меньшем количестве усилий ознакомиться с возможностями приложения и понять принципы работы в нём.

Под интерфейсом понимается любой экранный информационный или интерактивный интерфейс. Таковыми являются:

* сайты,
* мобильные приложения,
* приложения для стационарных компьютеров,
* презентационные панели,
* информационные стационарные экраны.

Проецируемая картинка на стену или полотно с использованием проектора и управляемая жестами или голосом тоже считается интерфейсом.

Этапы разработки

Полный цикл разработки интерфейса включает следующие этапы:

* Исследование
* Пользовательские сценарии
* Структура интерфейса
* Прототипирование интерфейса
* Определение стилистики
* Дизайн концепция
* Оформление всех экранов
* Анимация интерфейса
* Подготовка материалов для разработчиков

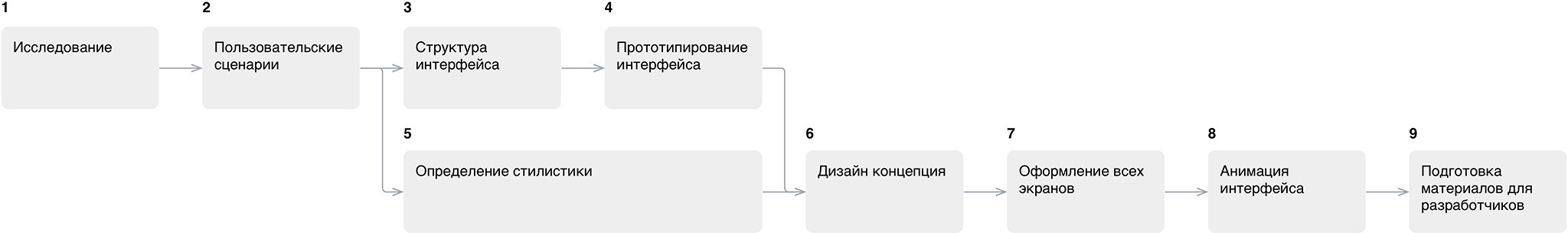


Рисунок 14 – Этапы разработки

**31.01.2020**

**Технология разработки программного обеспечения**

Процесс разработки программного обеспечения (англ. software development process, software process) — структура, согласно которой построена разработка программного обеспечения (ПО). Существует несколько моделей такого процесса, каждая из которых описывает свой подход, в виде задач и/или деятельности, которые имеют место в ходе процесса. Шаги процесса

Процесс разработки состоит из множества подпроцессов, или дисциплин, некоторые из которых показаны ниже. В модели водопада они идут одна за другой, в других аналогичных процессах их порядок или состав изменяется.

* Анализ требований;
* Спецификация программного обеспечения;
* Проектирование программного обеспечения;
* Программирование;
* Тестирование программного обеспечения ;
* Системная интеграция (System integration);
* Внедрение программного обеспечения (или Установка программного обеспечения);
* Сопровождение программного обеспечения.

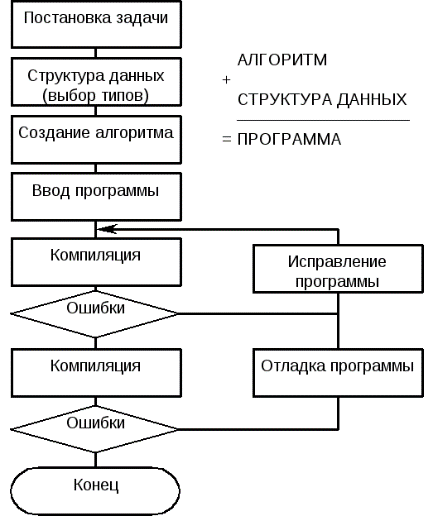


Рисунок 17 – Пример структуры программы

**Тестирование и отладка программного обеспечения**

Цели, с которыми производится тестирование программного обеспечения вполне очевидны. Первая задача, которая стоит перед командой испытателей – выполнить наглядную демонстрацию того, что продукт действительно работает. Это необходимо как самим разработчикам, так и заказчикам проекта. От вердикта последних зависит попадет программа к конечному потребителю, или нет. Вторая цель, реализации которой способствует тестирование – определение возможных изъянов разработанного продукта. Речь идет, в первую очередь, о ситуациях, в которых программное обеспечение может повести себя ненадлежащим образом. В ходе тестирования программного обеспечения, как и при любых других способах проверки продукции, определяется уровень качества ПО. Испытатели должны убедиться в том, что объект соответствует всем, предъявляемым к нему требованиям. Последние не очень многочисленны, но при этом, чрезвычайно конкретны. В первую очередь, оценивается функциональная пригодность продукта. О ней свидетельствует тот факт, что программа выполняет все функции, которые от нее требуются. Вторая характеристика, которая подлежит проверке – надежность. Тестирование позволяет выяснить, сможет ли программа выполнять все, возложенные на нее функции в заданных условиях. Третий фактор, который необходимо выяснить в ходе проверки – уровень производительности продукта. Здесь учитывается коэффициент полезного действия объекта исследования. Следующая характеристика, определяющая качество программного обеспечения – совместимость. Продукт должен работать на тех видах устройств, которые нужны заказчикам.

Не стоит забывать и об удобстве использования ПО. Любой, даже самый прогрессивный программный продукт, будет отвергнут, если конечному потребителю будет некомфортно с ним работать. В числе других важных характеристик, которые определяются в ходе тестирования программного обеспечения – мобильность, сопровождаемость и защищенность. Последний фактор особенно значим, так как именно он определяет способность продукта противостоять различным формам негативного воздействия.

**03.02.2020**

**Интеграция модуля в информационную систему**

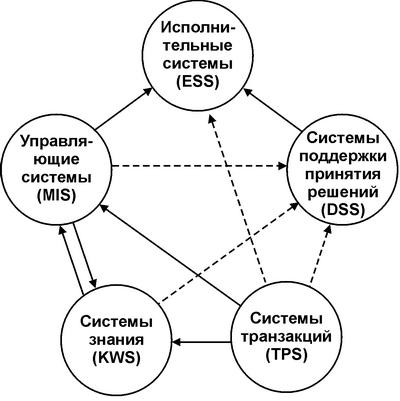
* Уровень данных

Создание дополнительного протокола или базы данных для удобного обмена информацией между приложениями, которые имеют различные форматы и типы данных.

* Программные и пользовательские интерфейсы

Разработка унифицированного интерфейса, реализующего прикладную бизнес-логику и позволяющего эффективно взаимодействовать со всеми компонентами, сохраняя неизменными слои обработки данных.

* Функционально-прикладной и организационный уровень

Консолидация однотипных функций и алгоритмов путем перераспределения потоков данных, перестройки организационных структур, а также модификации схемы информационного взаимодействия.

* Корпоративные программные приложения

Рисунок 18 – Интеграция модуля в ИС

Интеграция приложений осуществляется посредством включения дополнительных модулей, способных обращаться к нескольким системам, в том числе разными способами (например, через удаленный вызов процедур в одном случае и базу данных – в другом).

* Web-сервисы

Обеспечение доступа ко всем компонентам системы с использованием стандартного web-интерфейса.

Интеграция позволяет оптимизировать информационную систему за счет объединения приложений, используемых в рамках общего бизнес-процесса, но не связанных технологически.

В результате интеграции формируется цельная унифицированная экосистема приложений, позволяющая повысить эффективность работы, обеспечить удобство доступа к различным системам и значительно снизить затраты на внедрение новых программных решений и сервисов.

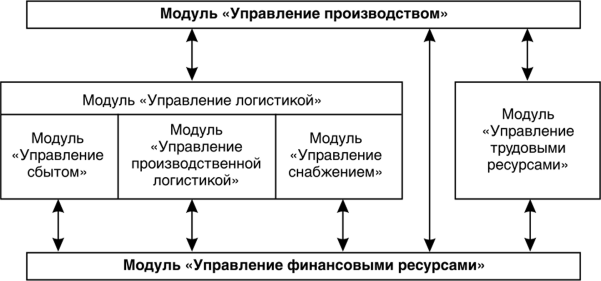


Рисунок 19 – Модули «Управление производством» и «Управление финансовыми ресурсами

**04.02.2020**

**Файловый и потоковый ввод-вывод**

Файловый и потоковый ввод-вывод относятся к передаче данных с носителя информации или на него. В .NET Framework пространства имен System.IO содержат типы, которые обеспечивают как синхронные, так и асинхронные операции чтения и записи для потоков и файлов. Кроме того, эти пространства имен содержат типы, выполняющие сжатие и распаковку файлов, а также типы, которые обеспечивают взаимодействие через каналы и последовательные порты.

Файл — это упорядоченная и именованная последовательность байтов, имеющая постоянное хранилище. При работе с файлами используются пути к каталогам, запоминающие устройства, а также имена файлов и каталогов. В отличие от файла, поток — это последовательность байтов, которую можно использовать для записи или чтения из вспомогательного запоминающего устройства, являющегося одним из устройств хранения информации (например, дисков или памяти). Есть несколько видов запоминающих устройств, отличных от дисков, и существует несколько видов потоков, помимо файловых потоков, например сетевые потоки, потоки памяти и потоки каналов.

**Файлы и каталоги**

Типы в пространстве имен System.IO можно использовать для взаимодействия с файлами и каталогами. Например, можно получать и задавать свойства файлов и каталогов, а также извлекать коллекции файлов и каталогов на основе критерия поиска.

Соглашения об именовании и способы указать путь к файлу в системах Windows, включая синтаксис устройств DOS, поддерживаемый в .NET Core 1.1 и более поздних версиях и платформе .NET Framework 4.6.2 и более поздних версиях.

Ниже перечислены некоторые часто используемые классы для файлов и каталогов:

* File предоставляет статические методы для создания, копирования, удаления, перемещения и открытия файлов, а также помогает создать объект FileStream.
* FileInfo предоставляет методы экземпляра для создания, копирования, удаления, перемещения и открытия файлов, а также помогает создать объект FileStream.
* Directory предоставляет статические методы для создания, перемещения и перечисления в каталогах и подкаталогах.
* DirectoryInfo предоставляет методы экземпляра для создания, перемещения и перечисления в каталогах и подкаталогах.
* Path предоставляет методы и свойства для обработки строк каталогов межплатформенным способом.

При вызове методов для работы с файловой системой следует всегда использовать надежные механизмы обработки исключений.

**Потоки**

Абстрактный базовый класс Stream поддерживает чтение и запись байтов. Все классы, представляющие потоки, являются производными от класса Stream. Класс Stream и его производные классы обеспечивают общий способ просмотра источников данных и хранилищ объектов, а также изолируют программиста от специфических особенностей операционной системы и базовых устройств.

Потоки включают три основные операции:

* Чтение — перенос информации из потока в структуру данных, такую как массив байтов.
* Запись — перенос данных в поток из источника данных.
* Поиск — определение и изменение текущей позиции внутри потока.

В зависимости от базового источника или хранилища данных поток может поддерживать лишь некоторые из этих возможностей. Например, класс PipeStream не поддерживает поиск. Свойства CanRead, CanWrite и CanSeek потока определяют операции, поддерживаемые потоком.

Ниже перечислены некоторые часто используемые классы потока:

* FileStream — для чтения и записи в файл.
* IsolatedStorageFileStream — для чтения и записи в файл в изолированном хранилище.
* MemoryStream — для чтения и записи в память в качестве резервного хранилища.
* BufferedStream — для повышения быстродействия операций чтения и записи.
* NetworkStream — для чтения и записи на сетевые сокеты.
* PipeStream — для чтения и записи в анонимные и именованные каналы.
* CryptoStream — для связи потоков данных с криптографическими преобразованиями.

**Средства чтения и записи**

Пространство имен System.IO также предоставляет типы для чтения закодированных символов из потоков и их записи в потоки. Как правило, потоки предназначены для ввода и вывода байтов. Типы чтения и записи обрабатывают преобразование закодированных символов в байты или из байтов, чтобы поток мог завершить операцию. Каждый класс чтения и записи связан с потоком, который можно получить с помощью свойства класса BaseStream.

Ниже перечислены некоторые часто используемые классы для чтения и записи:

* BinaryReader и BinaryWriter — для чтения и записи простых типов данных, таких как двоичные значения.
* StreamReader и StreamWriter — для чтения и записи символов с использованием закодированного значения для преобразования символов в байты или из байтов.
* StringReader и StringWriter — для чтения и записи символов в строки или из строк.
* TextReader и TextWriter используются в качестве абстрактных базовых классов для других средств чтения и записи, которые считывают и записывают символы и строки, а не двоичные данные.

**Асинхронные операции ввода-вывода**

Чтение и запись больших объемов данных может быть ресурсоемкой. Эти задачи необходимо выполнять асинхронно, если приложение должно продолжать отвечать на запросы пользователя. В случае синхронных операций ввода-вывода поток пользовательского интерфейса будет заблокирован до тех пор, пока ресурсоемкая операция не завершится. При разработке приложений Microsoft Store для Windows 8.x используйте асинхронные операции ввода-вывода, чтобы не создавалось впечатления, что приложение прекратило свою работу.

**Сжатие**

Сжатием называется процесс сокращения размера сохраняемого файла. Распаковка — это процесс извлечения содержимого сжатого файла, что приводит его в формат, пригодный для использования. Пространство имен System.IO.Compression содержит типы для сжатия и распаковки файлов и потоков.

При сжатии и распаковке файлов и потоков часто используются следующие классы:

* ZipArchive — для создания и восстановления содержимого ZIP-архива.
* ZipArchiveEntry — для представления сжатого файла.
* ZipFile — для создания, извлечения и открытия сжатого пакета.
* ZipFileExtensions — для создания и извлечения содержимого из сжатого пакета.
* DeflateStream — для сжатия и распаковки потоков с помощью алгоритма Deflate.
* GZipStream — для сжатия и распаковки потоков в формате gzip.

**Изолированное хранилище**

Изолированное хранилище — это механизм хранения данных, обеспечивающий изоляцию и безопасность путем определения стандартизованных способов сопоставления кода с хранимыми данными. Хранилище предоставляет виртуальную файловую систему, изолированную по пользователю, сборке и (необязательно) домену. Изолированное хранилище особенно полезно в том случае, когда приложение не имеет разрешения на доступ к файлам пользователя. Можно сохранить параметры или файлы для приложения таким способом, который контролируется политикой безопасности компьютера.

Изолированное хранилище недоступно для приложений Microsoft Store для Windows 8.x. Вместо этого используйте классы данных приложения в пространстве имен Windows.Storage. Дополнительные сведения см. в разделе [Данные приложения](https://docs.microsoft.com/previous-versions/windows/apps/hh464917%28v=win.10%29).

Часто используются следующие классы, реализующие изолированное хранилище:

* IsolatedStorage предоставляет базовый класс для реализации изолированного хранилища.
* IsolatedStorageFile предоставляет область изолированного хранилища, в которой содержатся файлы и каталоги.
* IsolatedStorageFileStream представляет файл в изолированном хранилище.

**Операции ввода-вывода в приложениях Microsoft Store**

.NET для приложений Магазина Windows 8.x содержит множество типов для чтения и записи в потоки, однако этот набор содержит не все типы ввода-вывода платформы .NET Framework.

Следует отметить некоторые важные различия в использовании операций ввода-вывода в приложениях Microsoft Store для Windows 8.x:

* Специальные типы, относящиеся к операциям с файлами, такие как File, FileInfo, Directory и DirectoryInfo, не включены в .NET для приложений Магазина Windows 8.x. Вместо этого используйте типы в пространстве имен Windows.Storage среды выполнения Windows, например StorageFile и StorageFolder.
* Изолированное хранилище недоступно; вместо этого используйте [данные приложения](https://docs.microsoft.com/previous-versions/windows/apps/hh464917(v=win.10)).
* Используйте асинхронные методы, такие как ReadAsync и WriteAsync, чтобы предотвратить блокировку потока пользовательского интерфейса.
* Типы сжатия на основе пути ZipFile и ZipFileExtensions недоступны. Вместо этого используйте типы в пространстве имен Windows.Storage.Compression.

При необходимости можно осуществлять преобразование между потоками .NET Framework и потоками среды выполнения Windows. Дополнительные сведения см. в разделе Практическое руководство. Преобразование между потоками .NET Framework и потоками среды выполнения Windows или WindowsRuntimeStreamExtensions.

**Ввод-вывод и безопасность**

При использовании классов в пространстве имен [System.IO](https://docs.microsoft.com/ru-ru/dotnet/api/system.io) необходимо выполнить требования безопасности операционной системы, такие как списки управления доступом для контроля доступа к файлам и каталогам. Это требование дополняет остальные требования FileIOPermission. Списками управления доступом можно управлять программно.

По умолчанию политика безопасности не позволяет обращаться к файлам на компьютере пользователя через Интернет или из приложений интрасети. Поэтому при составлении кода не используйте классы ввода-вывода, которым нужен путь к физическому файлу, загружаемому через Интернет или интрасеть. Вместо этого используйте [изолированное хранилище](https://docs.microsoft.com/ru-ru/dotnet/standard/io/isolated-storage) для обычных приложений .NET Framework или [данные приложения](https://docs.microsoft.com/previous-versions/windows/apps/hh464917(v=win.10)) для приложений Microsoft Store для Windows 8.x.

Проверка безопасности выполняется только при создании потока. Поэтому не рекомендуется открывать поток, а затем передавать его коду с меньшим уровнем доверия или доменам приложений.

**05.02.2020**

**Создание эмуляторов и подключение устройств**

**Эмуляторы**

Тестирование на физических устройствах – штука оправданная. Но это не значит, что не нужно тестировать на эмуляторах. Они позволяют расширить тестируемый набор устройств и проверять изменения на лету.

##### **Эмулятор Android**

У андроида эмулятор кросс-платформенный. К сожалению, он сложен в настройке.

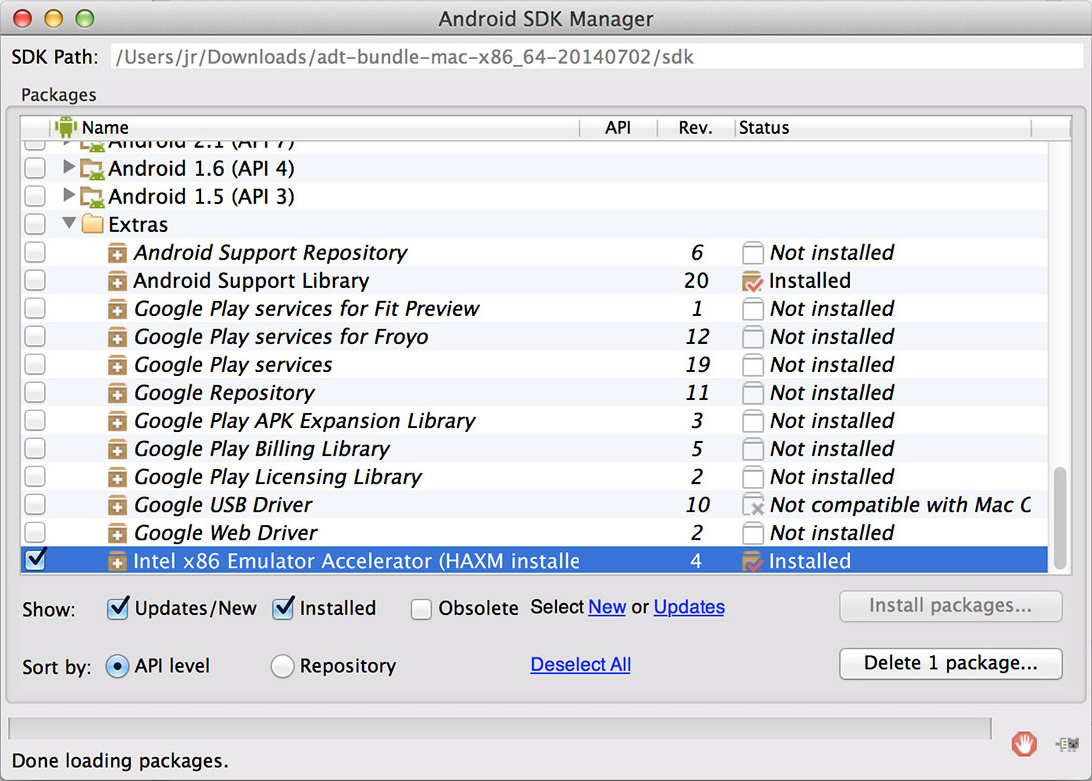
Для начала, скачайте подборку, включающую Android Development Tools (ADT) для Eclipse и Android software development kit (SDK). Затем следуйте инструкциям по установке, не забудьте кроме настроек по-умолчанию установить и “Intel x86 Emulator Accelerator (HAXM installer)”. Также вам понадобится установить HAXM (IntelHaxm.dmg на Маке и IntelHaxm.exe на PC)

Рисунок 20 – Andriod SDK Manager

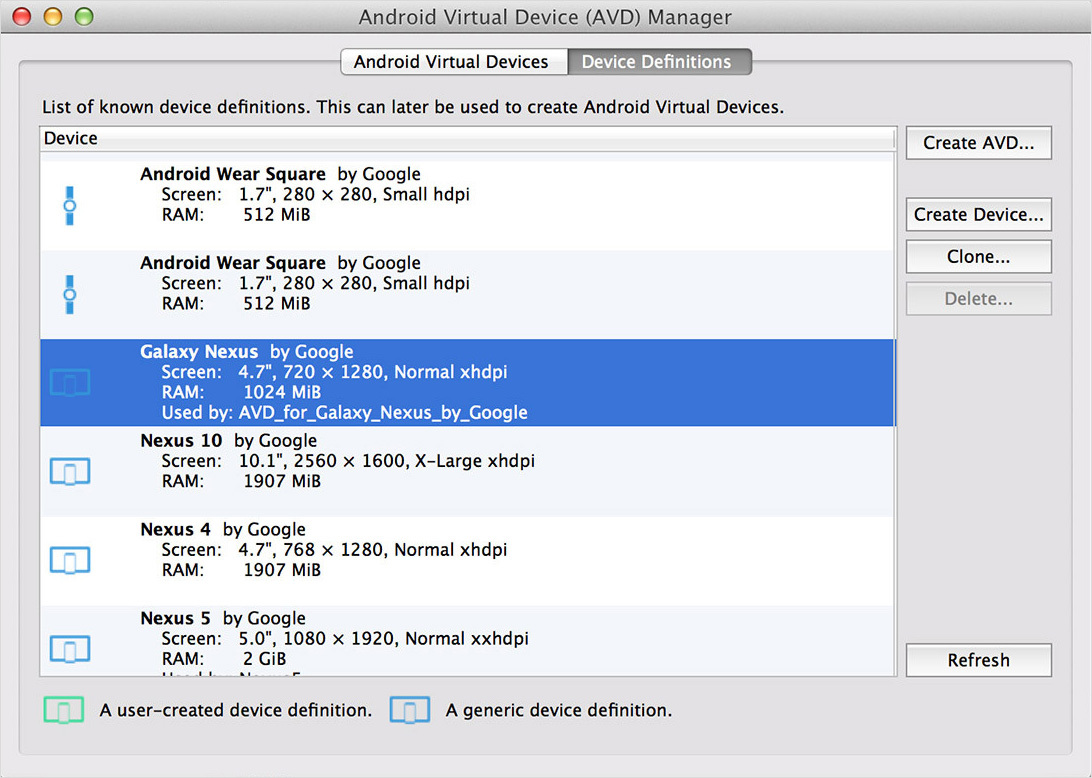
Затем создайте Android virtual device (AVD) для тестируемого устройства. В менеджере AVD есть список готовых устройств в “DeviceDefinitions”. Для начала выберите одно из них и нажмите “Create AVD”

Рисунок 21 – Andriod Virtual Device (AVD) Manager

Выберите любой CPU, и поставьте “No skin“ и “Use host GPU”. Теперь можно запускать виртуальное устройство и использовать браузер Android для тестирования.

Не помешает подучить [клавиатурные команды](http://developer.android.com/tools/help/emulator.html)для более удобного взаимодействия с эмулятором.

Как альтернативу, можно использовать [Manymo](https://www.manymo.com/)– эмулятор, встраиваемый в браузер. Его даже можно встроить в страницу.

Другие симуляторы и эмуляторы: [BlackBerry](http://us.blackberry.com/sites/developers/resources/simulators.html); [Windows Phone Emulator для Windows 8](http://msdn.microsoft.com/en-us/library/windows/apps/ff402563(v=vs.105).aspx);[Opera Mini Emulator](https://dev.opera.com/articles/installing-opera-mini-on-your-computer/)

**Подключение устройств**

**Android**

В случае с Андроидом инструменты удалённой отладки позволяют работать с десктопа при помощи Chrome’s Developer Tools. Кроме того, инструменты эти опять-таки кросс-платформенные.

##### Сначала, на телефоне пройдите в “Settings” → “About Phone” (для Android 4.4+), или “Settings” → “About Tablet”. Затем нужно нажать на “Build Number” семь раз. Кроме шуток. После этого вы увидите сообщение насчёт разработчика.

##### https://habrastorage.org/files/a62/f87/9b8/a62f879b89a7464bbb6a853249036355.jpgЗатем обратно в основных настройках “Developer Options” нужен пункт “USB debugging”.

Рисунок 22 – Android телефон

В адресной строке браузера Chrome наберите about:inspect. Разрешите “Discover USB devices”, и вы увидите в меню своё МУ.

Также вы должны увидеть открытые закладки в мобильном браузере. Выберите нужную, и вам будут доступны:

DOM Inspector,

Сетевая панель с внешними ресурсами

Панель исходного кода, для отладки JavaScript

Консоль JavaScript

Подробности можно найти в тьюториале “Introduction to Chrome Developer Tools, Part One.”

Рисунок 23 – Девайсы

Также можно заниматься удалённой отладкой в эмуляторе Андроида.

**06.02.2020**

**Тестирование и оптимизация мобильного приложения**

Тестирование – очень важный этап разработки мобильных приложений. Стоимость ошибки в релизе мобильного приложения высока. Приложения попадают в Google Play в течении нескольких часов, в Appstore несколько недель. Неизвестно сколько времени будут обновляться пользователи. Ошибки вызывают бурную негативную реакцию, пользователи оставляют низкие оценки и истерические отзывы. Новые пользователи, видя это, не устанавливают приложение. Мобильное тестирование сложный процесс: десятки различных разрешений экрана, аппаратные отличия, несколько версий операционных систем, разные типы подключения к интернету, внезапные обрывы связи.  
Поэтому в отделе тестирования у нас работает 8 человек (0,5 тестировщика на программиста), за его развитием и процессами следит выделенный тест-лид.

**Тестирование требований**

Тестирование начинается до разработки. Отдел дизайна передает тестировщикам навигационную схему и макеты экранов, менеджер проекта – требования невидимые на дизайне. Если дизайн предоставляет заказчик, макеты до передачи в отдел тестирования проверяются нашими дизайнерами.

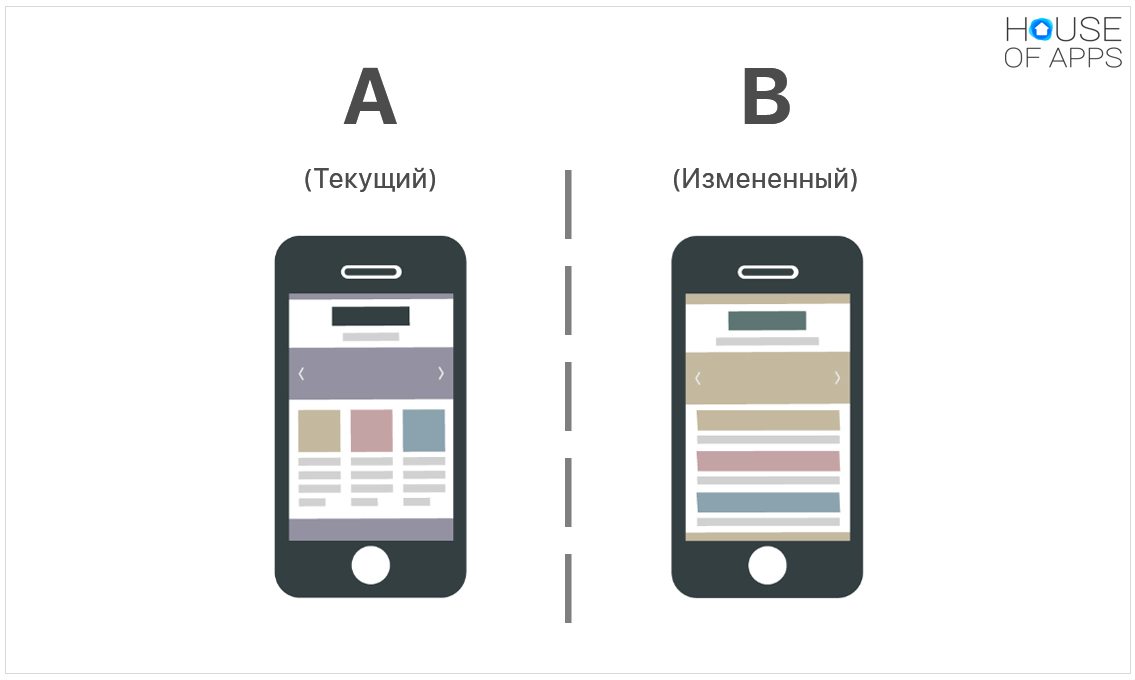
Тестировщик анализирует требования на полноту и противоречивость. В каждом проекте исходные требования содержат противоречивую информацию. Мы их решаем еще до начала разработки. Так же в каждом проекте требования неполны: не хватает макетов второстепенных экранов, ограничений на поля ввода, отображения ошибок, кнопки никуда не ведут. Неочевидны невидимые на макетах вещи: анимации, кеширование картинок и содержимого экранов, работа в нестандартных ситуациях. Недостатки требований обсуждаются с менеджером проекта, разработчиками и дизайнерами. После 2-3 итераций, вся команда гораздо лучше понимает проект, вспоминает забытый функционал, фиксирует решения по спорным вопросам. В основном на этом этапе используется basecamp. Когда требования стали полны и непротиворечивы, тестировщик составляет smoke-тесты и функциональные тесты, покрывающие исходные данные.

Рисунок 24 – Стадии тестирования

Если менеджер проекта поставит галочку «для тестирования», тестировщикам уходит письмо о новой сборке для тестирования. Ее номер отображается на мониторе в кабинете тестировщиков. Красным отображаются билды выпущенные за последние сутки, их нужно тестировать активнее, чем белые. Без «волшебного монитора» (кстати, работает на андроиде) часто тестировали старые билды. А новый билд с багами попадал заказчику. Теперь перед прогоном тест-кейсов достаточно взглянуть на монитор, путаница разрешилась. Тестирование билдов бывает быстрое и полное.

**03.04.2020**

**Подготовка отчета. Зачет.**

Во время практики я каждый день узнавал что-то новое. Каждый день я записывал в дневник практики пройденный за день материал, чтобы закрепить свои знания, а также использовать их при дальнейшей работе по специальности. Самое главное, правильно понять выданную тему и найти по ней информацию. Информации в интернете много и нужно правильно эту информацию фильтровать. В конце учебной практики необходимо было собрать все записи в единый отчет, а также дополнить дневник, вписав свои данные. Распечатать.

Также необходимо создать презентацию по темам, которые прошли за время учебной практики. Расписав слайды и добавив фотографии, относящиеся к темам, необходимо защититься.