Application Inputs and Outputs

Программа представляет собой механизм по переработке информации. На вход поступает информация в каком-то одном виде, на выходе – в другом. При этом входов и выходов у программы может быть много, они могут быть разными, т.е. у программы могут быть разные интерфейсы для общения с внешним миром, и эти интерфейсы могут иметь разные виды. Наиболее распространенные интерфейсы это:

- **пользовательский интерфейс (UI – User Interface)** - графический, текстовый (консольный), речевой и т.д. Через этот интерфейс компьютер взаимодействует с человеком;

- **программный интерфейс (API - Application Programming Interface)** - через этот интерфейс программы взаимодействуют друг с другом, а не с человеком;

- **сетевой протокол** – для взаимодействия программ друг с другом, но через сеть, а не непосредственно;

- **файловая система** – программы могут записывать данные на диск и читать данные с диска.

- **состояние окружения** – данные в окружении, которые программы могут использовать и/или модифицировать, например, переменные окружения в операционной системе и т.п.

- **события**, например, таймер, прерывание процессора и т.п.

Тестировщик создает искусственные ситуации и смотрит, как программа себя ведет на выходе при различных входных данных.

Схема тестирования примерно такова: Тестировщик получает программу и требования, после этого он с ними что-то делает, смотрит на поведение программы в определённых ситуациях и получает информацию о несоответствиях. Эта информация далее используется для того, чтобы улучшить программу, либо изменить требования, т. е. меняться может и то, и другое.

OS types

Program types

Software Architecture

Архитектура, строение, конфигурация программного обеспечения.

**Одно-ранговая архитектура** – приложение является вещью «само в себе»

**Клиент-сервер (**[**англ.**](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA)**Client-server)** — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемыми **серверами**, и заказчиками услуг, называемыми **клиентами**.

Бывают ситуации, когда и клиент, и сервер располагаются на одной машине, тогда такой сервер называют **локальным**.

Модель клиент-серверной архитектуры определяется распределением обязанностей между клиентом и сервером. Есть 3 уровня операций:

- уровень представления данных – это по сути интерфейс пользователя и отвечает за представление данных пользователя и ввод от него команд;

- прикладной уровень – реализует основную логику использования, на котором осуществляется обработка информации;

- уровень управления данными, который обеспечивает хранение данных и доступ к ним.

**Двухуровневая** архитектура предполагает взаимодействие 2-х программных модулей – клиентского и серверного. В зависимости от того, как между ними распределяются функции, различают:

- модель тонкого клиента – вся логика использования и управления данными сосредоточена на сервере. Клиентская программа обеспечивает функции только уровня представления;

- модель толстого клиента – сервер только управляет данными, а обработка информации и интерфейс сосредоточены на стороне клиента.

**Трёхслойная архитектура.** предполагает наличие следующих компонентов приложения:

* клиентское приложение (обычно говорят «[тонкий клиент](http://ru.wikipedia.org/wiki/%D0%A2%D0%BE%D0%BD%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82)» или терминал),
* сервер приложений,
* сервер [базы данных](http://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

**Пример 3-х звенной архитектуры клиент-сервер**

Компоненты трёхзвенной архитектуры, с точки зрения программного обеспечения реализуют определенные сервера БД, web-сервера и браузеры. Место любого из этих компонентов может занять программное обеспечение любого производителя. Ниже представлено описание взаимодействия компонентов трехуровневой архитектуры клиент-серверного приложения.

1. Сервер БД представлен Oracle-сервером;
2. сервер приложений технологиями: ADO.NET, ASP.NET и web-сервером IIS;
3. роль клиента выполняет любой web-браузер.

**Браузер клиента** *1->* **Сервер IIS** *2->* **Исполняющая среда ASP.NET 2.0** *3->* **Провайдер данных ADO.NET 2.0** *4->* **Сервер MySQL** *5->* **Провайдер данных ADO.NET 2.0** *6->* **Исполняющая среда ASP.NET 2.0** *7->* **Сервер IIS** *8->* Браузер клиента

* 1 — браузер клиента отправляет HTTP-запрос;
* 2 — на стороне сервера служба Web [Internet Information Services](http://ru.wikipedia.org/wiki/Internet_Information_Services) (web-сервер IIS) определяет тип запрашиваемого ресурса, и для случая запроса \*.aspx (расширение файлов страниц ASP.NET) загружает соответствующее ему (запросу) расширение Internet Server Application Programming Interface (ISAPI). Для страниц aspx это расширение isapi\_aspnet.dll. IIS также осуществляет идентификацию и авторизацию пользователя от которого поступил запрос. В свою очередь расширение isapi\_aspnet.dll загружает фабрику обработчиков ASP.NET. Далее, фабрика обработчиков создает объектную модель запрашиваемой страницы и обрабатывает действия пользователя.
* 3 — в ходе генерации ответа приложению ASP.NET может потребоваться обращение к БД, в этом случае используя библиотеки классов провайдера данных ADO.NET 2.0, выполняющая среда обращается к серверу БД;
* 4 — провайдер данных ADO.NET 2.0 передает запрос на операцию с БД серверу MySQL;
* 5 — сервер MySQL осуществляет обработку запроса, выполняя соответствующие операции с БД ;
* 6 — провайдер данных ADO.NET 2.0 передает результаты запроса объекту страницы;
* 7 — объект страницы с учетом полученных данных осуществляет рендеринг графического интерфейса страницы и направляет результаты в выходной поток;
* 8 — сервер IIS отправляет содержимое сгенерированной страницы клиентскому браузеру.

**Приемущества:**

* Делает возможным распределение функций вычислительной системы между несколькими независимыми компьютерами в сети. Это позволяет упростить обслуживание вычислительной системы. В частности, замена, ремонт, модернизация или перемещение сервера не затрагивают клиентов.
* На сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа. Все данные хранятся на сервере, который защищён гораздо лучше большинства клиентов.
* Позволяет объединить различные клиенты. Использовать ресурсы одного сервера часто могут клиенты с разными аппаратными платформами, операционными системами и т. п.

**Недостатки:**

* Неработоспособность сервера может сделать неработоспособной всю вычислительную сеть.
* Поддержка работы данной системы требует отдельного специалиста — системного администратора.
* Высокая стоимость оборудования.

## Software versioning

**Software versioning** — это процесс создания уникальных имен или номеров для различных версий продуктов программного обеспечения.

При имеющейся категории номера версии (главная, второстепенная), номера обычно выставляются в возрастающем порядке и соответствуют новым разработкам в программном обеспечении. На начальном уровне отслеживанием постепенно появляющихся версий электронной информации занимается система управления версиями, позволяющая хранить несколько версий одного и того же документа, при необходимости, возвращаться к более ранним версиям, определяя, кто и когда сделал то или иное изменение и многое другое. Вместе с тем для отслеживания изменений программного обеспечения было создано большое количество схем присвоения номеров версиям.

Каждой стадии разработки программного обеспечения соответствует уникальный идентификатор, который состоит из последовательности номеров или букв. В некоторых схемах последовательные идентификаторы используются для определения значимости перемен между стадиями разработки: эти перемены классифицируются по уровням значимости. Решение о том какую последовательность изменить между стадиями разработки основано на значимости перемен на последней стадии разработки, например в схеме, с версией, состоящей из последовательности 4 чисел, первое число может быть прибавлено только тогда когда код полностью переписан, в то время как четвертое число изменяется при незначительных переменах в интерфейсе или документации.

В более поздних релизах, главное число (major) увеличивается, когда происходят значительные переходы в функциональности, второстепенное число (minor) прибавляется только тогда, когда были добавлены незначительные функции или внесены исправления. Номер версии изменяется, если исправлены все мелкие неполадки. Для типичного программного продукта используются следующие номера: *0.9 (для бета-версии), 0.9.1, 0.9.2, 0.9.3, 1.0, 1.0.1, 1.0.2, 1.1, 1.1.1, 2.0, 2.0.1, 2.0.2, 2.1, 2.1.1, 2.1.2, 2.2*, и т.д. Разработчики порой перескакивают от версии 5.0 сразу к 5.5, для того чтобы обозначить добавление нескольких значимых функций в программе, однако их не достаточно, чтобы изменить главный номер версии, тем не менее такие скачки все же неуместны.

### Обозначение стадии разработки

   Для присвоения альфа или бета статуса релизам, которые еще не достаточно стабильны для практического применения, существуют схемы, где в первой последовательности используется ноль. Он ставится третьим числом в тех случаях, когда планируется еще тестировать версию или версия годна только для внутреннего применения.  
  
Последовательность в таких схемах следующая:

* **0** — альфа
* **1** — бета
* **2** — релиз кандидат
* **3** — публичный релиз

Например:

* **1.2.0.1** вместо **1.2-a**
* **1.2.1.2** вместо **1.2-b2** (бета с несколькими исправленными ошибками)
* **1.2.2.3** вместо **1.2-rc** (релиз кандидат)
* **1.2.3.0** вместо **1.2-r** (для коммерческого распространения)
* **1.2.3.5** вместо **1.2-r5** (для коммерческого распространения со многими исправленными ошибками)

### Приращение последовательности

   Существует два разных способа приращения последовательности номеров в версии. Большинство продуктов свободного программного обеспечения используют непрекращаемый поток последовательных номеров: *1.7.0*, *1.8.0*, *1.8.1*, *1.9.0*, *1.10.0*, *1.11.0*, *1.11.1*, *1.11.2*, и т.д. Примером такого продукта может служить [MediaWiki](http://en.wikipedia.org/wiki/MediaWiki). В других программах используются десятичные номера: *1.7*, *1.8*, *1.81*, *1.82*, *1.9*, и т.д. В таких программах после версии *1.8* будет идти версия *1.81*, текущие релизы будут обозначаться *1.81a*,*1.81b*, и т.д.

### Значимость нумерации версий в разработке программного обеспечения

   Номера версий используются в практических условиях потребителем или клиентом для того, чтобы можно было сравнить имеющуюся у них копию продукта программного обеспечения и новую версию, выпущенную разработчиком. Команда программистов и компании используют нумерацию версий для сравнения отдельных частей и секторов программного кода одних версий с другими, обычно сотрудничая с Системой контроля версий. Не существует абсолютной и определенной схемы нумерации версий продуктов программного обеспечения, поэтому очень часто нумерация зависит от личного выбора программистов.

[http://habrahabr.ru/blogs/development\_tools/118756/](http://habrahabr.ru/blogs/development_tools/118756/#habracut)

**Качество**

Стандарт ISO (International Organization for Standartization) 9126. Согласно этому стандарту можно выделить 6 аспектов качества, у которых можно выделить некоторое кол-во характеристик.

- функциональность;

- надежность (reliability);

- практичность;

- эффективность;

- сопровождаемость;

- переносимость.

Функциональность – основной аспект качества и наиболее важной его характеристикой является **пригодность к использованию (suitability)**. Т. е. программа должна делать то что она должна делать, для чего она предназначена.

Во-вторых она должна делать это правильно. Она должна вычислять все с определенной точностью, должна правильно конвертировать данные и т. д. **Точность, правильность (accuracy).**

Способность к взаимодействию с другими программами, с ОС, с другими версиями той же программы. **Способность к взаимодействию (interoperability)**

В частности она должна проддерживать какие-то стандарты и удовлетворять определенным правилам. **Соответствие стандартам и правилам (compliance).**

Интересной особенностью этого стандарта явл. **защищенность (security).** Этот стандарт не выделяется как аспект верхнего уровня, а считается подхарактеристикой функциональности. Пр-ма не должна разрушать данные, не должна мешать работеть другим программам, не должна предоставлять доступ к данным тем, кому этот доступ не разрешен.

Надежность:

**Зрелость (maturity)** которая является обратной величиной к частоте отказов.

**Устойчивость к отказам (fault tolerance)** – способность системы не реагировать на какие-то внутренние проблемы;

**Способнсоть к восстановлению работоспособности при отказах (recoverability)**. Если сервер упал, то система должна восстановиться, вернуть все необходимые данные, ничего не потерять и продолжить работу.

Практичность:

**Понятность (understandability)** пользователь должен понять ка воспользоваться программой для достижения своих целей;

**Удобство обучения (learnability)** – у прогарммы должна быть какая-то документация

**Работоспособность или управляемость (operability)** – пользователь должен иметь возможноть управлять поведением программы, она должна реагирвоать на его действия

Привлекательность (attractiveness) – эстетическая привлекательность.

Эффективность или производительность:

**Временные характеристики (time behavior)** – время отклика, скорость работы программы, скорость обработки определенных данных и т д.

**Использование ресурсов (resource utilization)** – использование дисковых ресурсов, использование процессора, оперативной памяти, сетевых ресурсов.

Сопровождаемость:

Этот аспект важен не сколько юзерам, сколько самим разработчикам, поэтому является внутренним аспектом, а не внешним.

**Анализируемость (analyzability)** -

**Изменяемость, удобство внесения изменений (changeability)**

риск возникновения неожиданных эффектов при внесении изменений **(stability)**

контролируемость или удобство тестирования программы **(testability)**

Переносимость или мобильность:

Пр-ма должна работать в различных окружениях, быть достаточно проста в установке, должна уменять работать с другими программами и не мешать им и она не должны мешать ей должна обеспечивать пользователь легко перейти от использования другого ПО на эту программу, т. е. предоставить какие-то возможности по миграции, чаще всего это миграции данных.

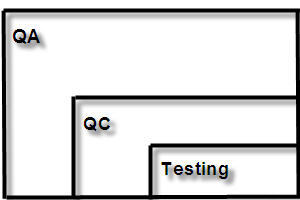
**Адаптируемость (adaptability)**

**Устанавливаемость, удобство установки (installability)**

**Способность к сосуществованию с другими ПО (coexistence)**

**Удобство замены другого ПО данным (replaceability)**

В соответствии с данными аспектами качества можно выделить аналогичные виде тестирования.



**Testing**(тестирование) - это действия по проверке програмного продукта, прохождение тест кейсов и локализация дефектов, минимальный необходимый уровень знаний и умений   
**QC**- следующий уровень - контроль качества продукта - анализ результатов тестирования и качества “билдов”, в процессе разработки.  
**QA**- решает более глобальные задачи. Анализируя работу тестировщиков и QC, в случае возникновения проблем, вовремя находит пути ее решения и не дает ей развиться и повлиять на качество продукта.

**QA** (**Quality Assurance**, **Обеспечение качества**) - совокупность мероприятий, охватывающих все технологические этапы разработки, выпуска и эксплуатации ПО информационных систем, предпринимаемых на разных стадиях **жизненного цикла ПО**, для **обеспечения качества выпускаемого продукта**.  
**QC** (**Quality Control**, **Контроль качества**) - совокупность действий проводимых над объектом тестирования в процессе разработки для получения информации об актуальном состоянии объекта тестирования в разрезах: "**готовность Продукта к выпуску**", "**Соответствие зафиксированным требованиям**", "**Соответствие заявленному уровню качества продукта**".  
**Software Testing** является одной из техник контроля качества и включает в себя активности по планированию работ (**Test Management**), проектированию тестов (**Test Design**), выполнению тестирования (**Test Execution**) и анализу полученных результатов (**Test Analysis**).

<http://alexeybulat.blogspot.com/2007/12/qa-qc-testing.html> - продолжение здесь

## PDS responsibilities

Product specialists and Business Analysts – persons, who communicate directly with clients, discuss and interpretate there needs to functional requirements that are base for creating issues and CRs for developers and QA.

PDS (product specialist) responsibilities includes strong understanding of product functionality and features to participate in discussions and making decisions on covering and fulfilling requirements and needs of a client. To recognize whether particular case is defect or specific feature, to achieve concurrency with clients on how specific piece of functionality should work.

## BA responsibilities

BA (Business analyst) responsibilities includes all product specialists responsibilities and additionally clear vision of product features to have ability to convince clients that current problem is some specific feature or it is result of not correct usage or setup, etc. Also convince client that all his needs are fulfilled in particular product and he doesn’t need anything else, writing functional requirements.

## Working with requirements

BA, after concurrency with client is achieved, is writing functional requirements that should be implemented in next versions of product by developers and covered by test cases and after that tested. Than BA reviews that functionality and cases. If everything is ok, he makes meeting with PDA and clients (usually it is a conference call) and presents changes to the client. Client makes decision whether this solution fulfills his goals, and approves or rejects it.

<http://nrukol.blogspot.com/2010/05/qa-qc-testing-process-managers.html> - QA, QC & Testing

[http://www.softwaretestinghelp.com](http://www.softwaretestinghelp.com/) – хороший сайт про тестинг