Тестирование и тест-дизайн. Основы функционального тестирования. Модульные тесты.

1.Тестирование и тест-дизайн.

- Любое тестирование, в том числе тестирование ПО - это поиск багов.
- Баг - это отклонение фактического результата (неких действий) от ожидаемого.

Чтобы проводить тестирование, нужно:

- 1. Узнать ожидаемый результат
- 2. Узнать фактический результат
- 3. Сравнить эти результаты

- *Например, нам нужно протестировать пуленепробиваемое стекло. Ожидается, что его нельзя пробить пулей. Чтобы протестировать, мы должны попытаться опровергнуть это ожидание, то есть выстрелить в стекло и узнать фактический результат. Затем сравним его с ожидаемым: если пуля пробила стекло, значит, найден баг.*

Таким образом, для тестирования нужно еще выбрать некоторое действие (в данном случае - выстрел), получаем, что процесс тестирования состоит из четырех стадий:

- 1. Выбрать действие
- 2. Узнать ожидаемый результат этого действия
- 3. Узнать фактический результат
- 4. Сравнить эти результаты

Первые две стадии относятся к тест-дизайну - написанию тестов.

- 1. Выбор действия (тестового сценария).
- Если у нас есть официальный документ с требованиями к продукту (спецификация), то тестовые сценарии следует написать, исходя из этих требований.

В хорошей спецификации основные сценарии использования (use-cases) прописаны явно, в виде пошаговых инструкций, которые можно использовать при тестировании "как есть".

Если спецификация не содержит пошаговых инструкций, а лишь общие слова, дизайнер тестов должен написать такие инструкции сам.

- Для более тщательного тестирования можно придумать свои тестовые сценарии. Существует много разных техник выбора таких сценариев.

- 2. Информацию об ожидаемом результате, опять же, правильнее всего взять из спецификации.

Если же в требованиях это не описано, можно использовать :

- 2.1. Авторитетное мнение (правильнее всего - того человека, который составляет спецификации)
- 2.2. Устоявшиеся стандарты (официальные, например, RFC, или стандарты де-факто, например, то, что при нажатии правой кнопки мыши появляется контекстное меню)
- 2.3. Здравый смысл
- 2.4. Заглянуть в код программы
- 2.5. Прочее

Последние две стадии - это собственно тестирование (прохождение тестов):

- 3. Узнать фактический результат мы можем, произведя действие, выбранное на первом этапе.
- 4. После этого нам остается сравнить фактический результат с ожидаемым и зарепортить баг в случае несоответствия.

Например, проведем тестирование электрического чайника.

- 1. Какое действие покажет нам, работает ли чайник? Самое очевидное - включить чайник.
- 2. Что мы примем за ожидаемый результат? Вероятно, то, что вода в чайнике через определенное время вскипит.
- 3. Как мы узнаем фактический результат? Включим чайник и подождем определенное время.
- 4. А теперь сравним фактический результат с ожидаемым. Здесь важен вопрос - как понять, что вода вскипела? Нужен критерий соответствия фактического результата и ожидаемого.
- Критерием может быть, например:
- а) Вода бурлит. Минус этого критерия - его нечеткость. Что значит "бурлит"? Как сильно должна вода бурлить? Кроме того, критерий пригоден только для тестирования с участием человека и практически непригоден для автоматического тестирования. Основываясь на этом критерии, сложно будет сделать чайник, который сам умеет отключаться.
- б) Температура воды достигла 99 градусов. Это уже более четкий критерий.

Если мы ждали достаточно долго, а вода так и не закипела, это означает, что мы нашли баг.

- Для тестировщика найденный баг - это всегда праздник, потому что это значит, что результат работы можно предъявить начальству.
- Не так важно количество найденных багов, как их серьезность (severity). Серьезность определяется тестовым сценарием, по результату которого был найден баг.

Тестовая документация.

- Тестовая документация состоит обычно из отдельных сценариев, которые называются тест-кейсами и могут быть для удобства объединены в группы.
- Написание тестовой документации имеет много общего с написанием ПО: следует разбивать код на отдельные модули и избегать дублирования кода.

2.1. Что должна содержать тестовая документация и почему.

- заголовок,
- пошаговое описание,
- ожидаемый результат,
- критерий соответствия ожидаемого результата фактическому.

- Пример.
- Заголовок: "Проверка того, что программа умеет показывать файлы формата BMP"
- Шаг 1. Нажать кнопку "Выбрать файл"
- Шаг 2. Выбрать файл с расширением BMP
- Шаг 3. Нажать кнопку "Открыть"
- Ожидаемый результат: содержимое файла показано в графическом виде, в полноэкранном режиме.
- Здесь сравнение ожидаемого результата и фактического осуществить довольно просто, и критерий соответствия не нужен. Приведем более сложный пример:

Жизненный цикл бага.

- Итак, тесткейсы написаны, назначены тестировщикам и пройдены. Тестировщик нашел некоторые баги при прохождении тесткейсов, занес их в систему учета багов (bug tracking system), например в Bugzilla, и пометил тесткейсы в системе тестовой документации как
- 1) пройденные успешно,
- 2) пройденные с багами, или
- 3) заблокированные (невозможно пройти из-за более общих багов).

- Таким образом, баги чинятся не всегда. А если чинятся, то в большинстве случаев тестировщик должен проверить, действительно ли баг починили.
- Жизнь бага представляет собой довольно сложный и разветвленный процесс. Чтобы было удобно этот процесс контролировать, в системе учета багов каждый баг имеет некий статус (состояние) и назначенного человека, который должен заниматься этим багом.
- Баг может переходить из одного состояния в другое и от одного человека к другому, пока не будет закрыт.
- В разных проектах жизненный цикл бага и список таких состояний различен, но в основном используются следующие состояния:

- NEW - баг только что найден
- ASSIGNED - назначенный человек начал заниматься этим багом
- RESOLVED - проблема решена (баг разрезолвен), это состояние имеет несколько вариантов
- FIXED - баг починили
- DUPLICATE - такой баг уже был занесен в систему учета,
- INVALID - такое поведение системы является ожидаемым
- WORKSFORME - баг не удалось воспроизвести
- WONTFIX - баг признали багом, но фиксить не будут. такая ситуация, как правило, возникает в случае несущественного бага, требующего больших трудозатрат на починку, точнее в случае слишком большого соотношения затрат к серьезности бага (цена/качество). Такую резолюцию вправе устанавливать на баг, как правило, только руководитель разработчиков или руководитель проекта.

Основы функционального тестирования (Black-Box)

1. Определение

Функциональное тестирование - это проверка ПО на правильность функционирования в идеальных условиях, в отличие от нефункционального, где либо условия неидеальны (нагрузочное тестирование), либо тестируется не правильность функционирования(тестирование удобства пользования или структура программы).

Как правило, функциональное и нефункциональное тестирование ПО можно проводить параллельно, поэтому обычно это делается разными людьми или командами.

В большинстве источников указывается, что функциональное тестирование - это синоним black-box тестирования (при котором программа рассматривается как черный ящик).

Black-box, white-box, grey-box тестирование.

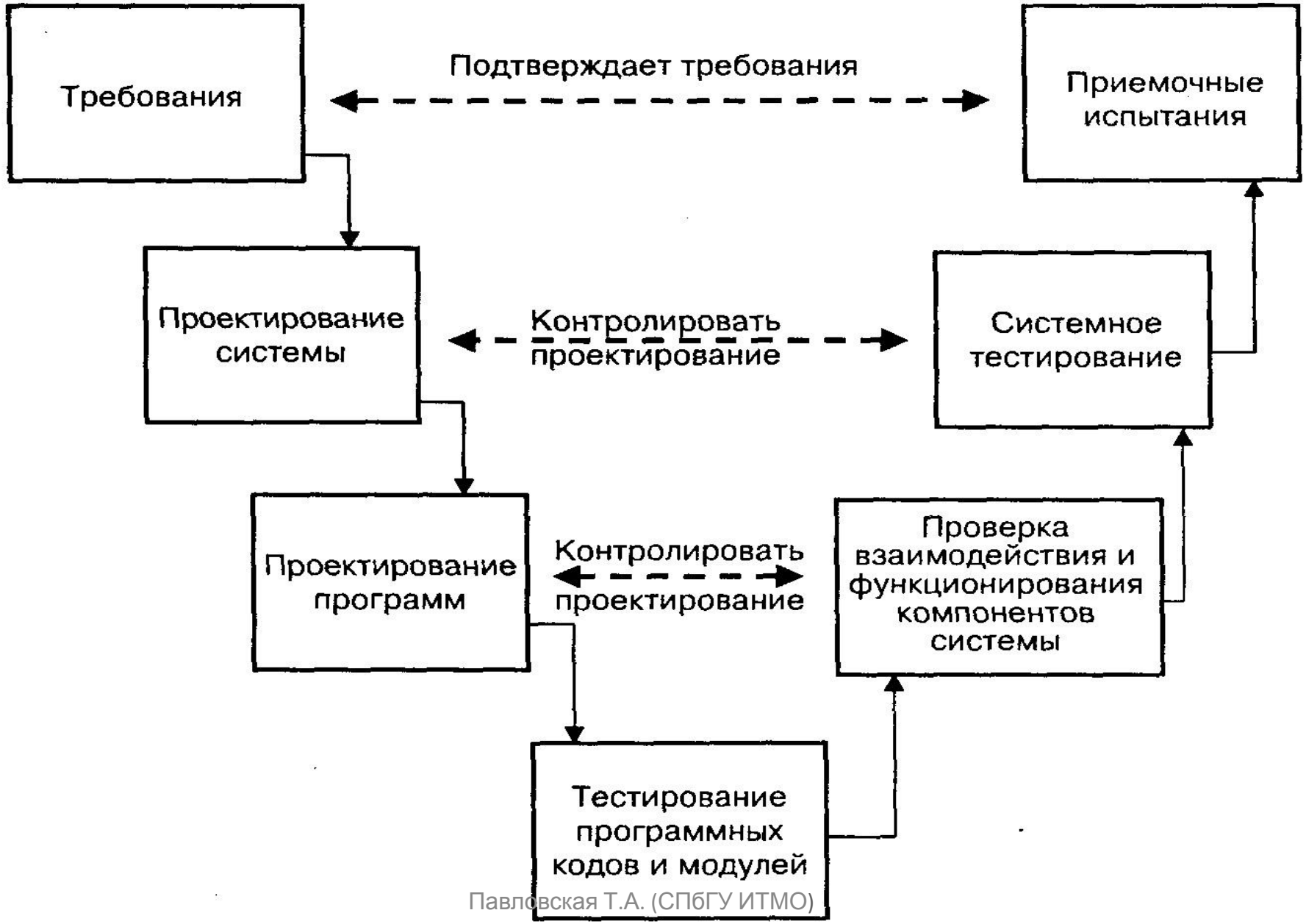
- По степени доступа к коду различают два типа тестирования: тестирование "черного ящика" (black box), или функциональное тестирование - без доступа к коду, и "белого ящика" (white box), или структурное тестирование - тестирование кода.
- В случае "черного ящика" программа рассматривается как конечный автомат, с входными и выходными данными, набором внутренних состояний и переходов между ними.
- В случае "белого ящика" тестировщик пишет тесткейсы, основываясь исключительно на коде программы (тесты на правильность кода).
- Расширение black-box тестирования, в котором также применяется изучение кода, называется тестированием "серого ящика" (grey box).

Тестирование сценариев использования - юз-кейсов (use-cases)

- Чтобы уменьшить число тестов, можно проверить только те переходы, которые имеют смысл для пользователя.
- Use-case - это логически завершенная последовательность действий.
- Тестирование сценариев является самым необходимым видом тестирования. Программа должна выполнять операции, для которых она предназначена. Если пользователь может выбрать пункт меню, но файл не открывается - это очень серьезный баг.

- Главным минусом Black-box тестирования является то, что тестировщик не знает, какую часть ПО он тестирует. Некоторые существующие пути в программе (о которых нет информации ни в требованиях, ни в документации), могут никогда не быть проверены. Уменьшить количество таких путей можно путем анализа внутреннего устройства программы.

Взаимосвязь разработки и тестирования (V-диаграмма)



Модульное тестирование (Unit testing)

- ***Модульное тестирование*** - это тестирование программы на уровне отдельно взятых модулей, функций или классов.
- Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования.
- Модульное тестирование чаще всего проводится по принципу "белого ящика“.

Обнаруживаемые ошибки

- На уровне модульного тестирования проще всего обнаружить дефекты, связанные с алгоритмическими ошибками и ошибками кодирования алгоритмов.
- Ошибки, связанные с неверной трактовкой данных, некорректной реализацией интерфейсов, совместимостью, производительностью и т.п. обычно выявляются на более поздних стадиях тестирования.