**Лекция 4**

**Особенности индустриального тестирования**

**Основные вопросы**

1. Издержки тестирования.
2. Качество программного продукта.
3. Фазы процесса тестирования.
4. Планирование тестирования.
5. Типы тестирования.
6. Подходы к разработке тестов.
7. Документация и сопровождение тестов.
8. Оценка качества тестов

### **Качество программного продукта и тестирование**

*Качество программного продукта* можно оценить некоторым набором характеристик, определяющих, насколько продукт "хорош" с точки зрения всех потенциально заинтересованных в нем сторон. Такими сторонами являются:

* заказчик продукта
* спонсор
* конечный пользователь
* разработчики продукта
* тестировщики продукта
* инженеры поддержки
* отдел обучения
* отдел продаж и т.п.

Каждый из участников может иметь различное *представление* о продукте и по-разному судить о том, насколько он хорош или плох, то есть насколько высоко качество продукта. С точки зрения разработчика, продукт может быть настолько хорош, насколько хороши заложенные в нем алгоритмы и технологии. Пользователю продукта, скорее всего, безразличны детали внутренней реализации, его в первую *очередь* волнуют вопросы функциональности и надежности. Спонсора интересует цена и совместимость с будущими технологиями. Таким образом, задача обеспечения качества продукта выливается в задачу определения заинтересованных лиц, согласования их критериев качества и нахождения оптимального решения, удовлетворяющего этим критериям.

В рамках подобной задачи *группа* тестирования рассматривается не просто как еще одна заинтересованная сторона, но и как сторона, способная оценить удовлетворение выбранных критериев и сделать *вывод* о качестве продукта с точки зрения других участников. К сожалению, далеко не все критерии могут быть оценены группой тестирования. Поэтому ее внимание в основном сосредоточено на критериях, определяющих *качество программного продукта* с точки зрения конечного пользователя.

**Тестирование как способ обеспечения качества**. Тестирование, с технической точки зрения, есть процесс выполнения приложения на некоторых входных данных и проверка получаемых результатов с целью подтвердить их *корректность* по отношению к результату.

Тестирование не позиционируется в качестве единственного способа обеспечения качества. Оно является частью общей системы обеспечения качества продукта, элементы которой выбираются по критерию наибольшей эффективности применения в конкретном проекте.

Рассмотрим пример. В качестве приложения возьмем программу для работы с сетью (*browser*), критерии качества которой приведены в [Табл.9.1](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=1#table.9.1).

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 9.1. Критерии качества программы browser | | | |
|  | **Пользователь** | **Заказчик** | **Инженер поддержки** |
| ***Функциональная полнота*** | + | - | - |
| **Цена разработки** | - | + | - |
| **Отсутствие дефектов** | + | Косвенно | + |
| **Удобство использования** | + | - | - |
| **Возможность внесения изменений в будущем** | - | Косвенно | + |
| **Легкость исправления дефектов** | - | - | + |
| **Документация на реализацию, в том числе комментарии** | - | - | + |
| **Своевременность исполнения проекта** | - | + | - |

*Матрица* критериев качества заинтересованных в них участников для рассматриваемого проекта приведена в [таблице 9.2](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=1#table.9.2) . Допустим, что вид матрицы критериев качества и проверяющих элементов системы обеспечения качества для данного проекта будет следующим:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица 9.2. Матрица критериев качества и элементов системы обеспечения качества | | | | | |
|  | **Тестирование** | **Анализ рынка и специальные лаборатории1** | **Обзоры кода** | **Анализ дизайна** | **Аудиты процесса разработки** |
| **Полнота функциональности** | +, не всегда эффективно | + | - | - | - |
| **Стоимость разработки** | - | - | - | - | + |
| **Отсутствие дефектов** | + | - | + | - | - |
| **Удобство использования** | +, не всегда эффективно | + | - | - | - |
| **Возможность внесения изменений в будущем** | - | - | +- | + | - |
| **Легкость исправления дефектов** | - | - | + | + | - |
| **Документация на реализацию, в том числе комментарии** | - | - | + | - | + |
| **Своевременность исполнения проекта** | - | - | - | - | + |

Данные ([Табл. 9.1](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=1#table.9.1), [Табл. 9.2](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=1#table.9.2)) показывают, что из восьми элементов общего качества продукта тестирование способно оценить и контролировать только три (1, 3, 4), причем наиболее эффективно тестирование контролирует отсутствие дефектов (3).

В каждом конкретном проекте элементы системы должны быть выбраны так, чтобы обеспечить приемлемое качество, исходя из приоритетов и имеющихся ресурсов. Выбирая элементы для системы обеспечения качества конкретного продукта, можно применить комбинированное тестирование, обзоры кода, *аудит*. При подобном выборе некоторые качества, например легкость модификации и исправления дефектов, не будут оценены и, возможно, выполнены. Задачей тестирования в рассматриваемом случае будет обнаружение дефектов и оценка удобства использования продукта, включая полноту функциональности. Исходя из задач, поставленных перед группой тестирования в конкретном проекте, выбирается соответствующая стратегия тестирования. Так, в данном примере, ввиду необходимости оценить *удобство использования* и полноту функциональности, преимущественный подход к разработке тестов следует планировать на основе использования сценариев.

Итак, основная последовательность действий при выборе и оценке критериев *качества программного продукта* включает:

1. Определение всех лиц, так или иначе заинтересованных в исполнении и результатах данного проекта.
2. Определение критериев, формирующих представление о качестве для каждого из участников.
3. Приоритезацию критериев, с учетом важности конкретного участника для компании, выполняющей проект, и важности каждого из критериев для данного участника.
4. Определение набора критериев, которые будут отслежены и выполнены в рамках проекта, исходя из приоритетов и возможностей проектной команды. Постановка целей по каждому из критериев.
5. Определение способов и механизмов достижения каждого критерия.
6. Определение стратегии тестирования исходя из набора критериев, попадающих под ответственность группы тестирования, выбранных приоритетов и целей

### **Процесс тестирования**

Как отмечалось в подразделе 2.4, в тестировании выделяются три основных уровня, или три фазы:

1. *Модульное тестирование*.
2. *Интеграционное тестирование*.
3. *Системное тестирование*.

Задача планирования активности тестирования состоит в оптимальном распределении ресурсов между всеми типами тестирования. В дальнейшем изложении мы сконцентрируемся на системной *фазе тестирования*, как на наиболее важной и критичной активности для разработки качественного программного продукта.

#### Фазы процесса тестирования

В процессе тестирования выделяют следующие фазы:

1. **Определение целей** (требований к тестированию), включающее следующую конкретизацию: какие части системы будут тестироваться, какие аспекты их работы будут выбраны для проверки, каково желаемое качество и т.п.
2. **Планирование**: создание графика (расписания) разработки тестов для каждой тестируемой подсистемы; оценка необходимых человеческих, программных и аппаратных ресурсов; разработка расписания *тестовых циклов*. Важно отметить, что расписание тестирования обязательно должно быть согласовано с расписанием разработки создаваемой системы, поскольку наличие исполняемой версии разрабатываемой системы ( **Implementation Under Testing (IUT)** или **Application Under Testing (AUT)** – часто употребляемые обозначения для тестируемой системы) является одним из необходимых условий тестирования, что создает взаимозависимость в работе команд тестировщиков и разработчиков.
3. **Разработка тестов,** то есть тестового кода для тестируемой системы, если необходимо - кода системы *автоматизации тестирования* и *тестовых процедур* (выполняемых вручную).
4. **Выполнение тестов**: реализация *тестовых циклов*.
5. **Анализ результатов**.

После анализа результатов возможно повторение процесса тестирования, начиная с пунктов 3, 2 или даже 1.

#### Тестовый цикл

***Тестовый цикл*** – это цикл исполнения тестов, включающий фазы 4 и 5 тестового процесса. *Тестовый цикл* заключается в прогоне разработанных тестов на некотором однозначно определяемом срезе системы (состоянии кода разрабатываемой системы). Обычно такой срез системы называют **build**. *Тестовый цикл* включает следующую последовательность действий:

1. Проверка готовности системы и тестов к проведению *тестового цикла* включающая:
   * Проверку того, что все тесты, запланированные для исполнения на данном цикле, разработаны и помещены в систему версионного контроля.
   * Проверку того, что все подсистемы, запланированные для тестирования на данном цикле, разработаны и помещены в систему версионного контроля.
   * Проверку того, что разработана и задокументирована процедура определения и создания среза системы, или build.
   * Проверки некоторых дополнительных критериев.
2. **Подготовка тестовой машины** в соответствии с требованиями, определенными на этапе планирования (например, полная очистка и переустановка системного программного обеспечения). Конфигурация тестовой машины, так же, как и срез системы, должны быть однозначно воспроизводимыми.
3. **Воспроизведение среза системы**.
4. **Прогон тестов** в соответствии с задокументированными процедурами.
5. **Сохранение тестовых протоколов (test log)**. *Test log* может содержать вывод системы в STDOUT, список результатов сравнения полученных при исполнении данных с эталонными или любые другие выходные данные тестов, с помощью которых можно проверить правильность работы системы.
6. **Анализ протоколов тестирования** и принятие решения о том прошел или не прошел каждый из тестов ( **Pass/Fail** ).
7. **Анализ и документирование** результатов цикла.

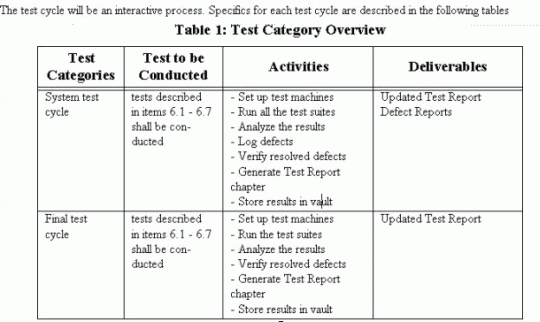
Последний перед выпуском продукта *тестовый цикл* не должен включать изменений кода build или кода продукта тестируемой системы. Этот цикл называется " финальным ". Таким образом обеспечивается ситуация, когда финальный цикл полностью повторяем, а выпускаемый продукт полностью совпадает с продуктом, который прошел тестирование. Финальный цикл необходим для гарантии достоверности результатов тестирования.

### Планирование тестирования

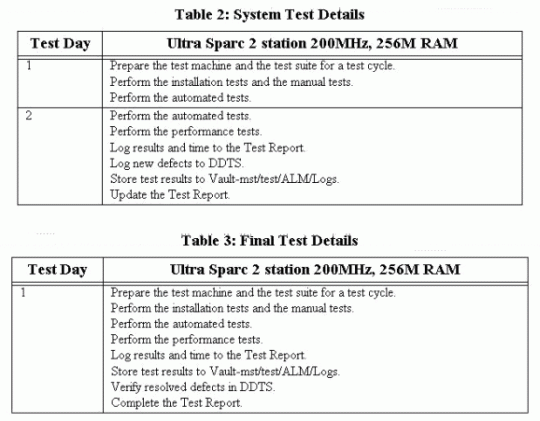
#### Тестовый план

***Тестовый план*** - это документ, или набор документов, содержащий следующую информацию:

1. Тестовые ресурсы.
2. Перечень функций и подсистем, подлежащих тестированию.
3. *Тестовую стратегию*, включающую:
   * Анализ функций и подсистем с целью определения наиболее слабых мест, то есть областей функциональности тестируемой системы, где появление дефектов наиболее вероятно.
   * Определение стратегии выбора входных данных для тестирования. Так как множество возможных входных данных программного продукта, как правило, практически бесконечно, выбор конечного подмножества, достаточного для проведения исчерпывающего тестирования, является сложной задачей. Для ее решения могут быть применены такие методы, как покрытие классов входных и выходных данных, анализ крайних значений, покрытие модели использования, анализ временной линии и тому подобные. Выбранную стратегию необходимо обосновать и задокументировать.
   * Определение потребности в *автоматизированной системе тестирования* и дизайн такой системы
4. Расписание *тестовых циклов* (пример приведен на [Рис. 9.1](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=3#image.9.1)).
5. Фиксацию тестовой конфигурации: состава и конкретных параметров аппаратуры и программного окружения (пример приведен на [Рис. 9.2](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=3#image.9.2)).
6. Определение списка тестовых метрик, которые на *тестовом цикле* необходимо собрать и проанализировать. Например, метрик, оценивающих степень покрытия тестами набора требований, степень покрытия кода тестируемой системы, количество и *уровень серьезности* дефектов, объем тестового кода и другие характеристики.



**Рис. 9.1.**Пример расписания двух последних тестовых циклов



**Рис. 9.2.**Пример детализации условий проведения системных циклов

#### Типы тестирования

В *тестовом плане* определяются и документируются различные типы тестов. Типы тестов могут быть классифицированы по двум категориям: по тому, что подвергается тестированию (по виду подсистемы) и по способу выбора входных данных.

Типы тестирования по виду подсистемы или продукта:

1. **Тестирование основной функциональности**, когда тестированию подвергается собственно система, являющаяся основным выпускаемым продуктом
2. **Тестирование инсталляции** включает *тестирование сценариев* первичной инсталляции системы, сценариев повторной инсталляции (поверх уже существующей копии), тестирование деинсталляции, тестирование инсталляции в условиях наличия ошибок в инсталлируемом пакете, в окружении или в сценарии и т.п.
3. **Тестирование пользовательской документации** включает проверку полноты и понятности описания правил и особенностей использования продукта, наличие описания всех сценариев и функциональности, синтаксис и грамматику языка, работоспособность примеров и т.п.

Типы тестирования по способу выбора входных значений:

1. **Функциональное тестирование**, при котором проверяется:
   * Покрытие функциональных требований.
   * Покрытие сценариев использования.
2. **Стрессовое тестирование**, при котором проверяются экстремальные режимы использования продукта.
3. **Тестирование граничных значений**.
4. **Тестирование производительности**.
5. **Тестирование на соответствие стандартам**.
6. **Тестирование совместимости** с другими программно-аппаратными комплексами.
7. **Тестирование работы с окружением**.
8. **Тестирование работы на конкретной платформе**

В реальных разработках используются и комбинируются различные типы тестов для обеспечения спланированного качества продукта.

### Подходы к разработке тестов

Рассмотрим разные подходы к разработке тестов, два к выбору тестовых данных и два к реализации тестового кода.

#### Тестирование спецификации

При разработке тестов, основанных на *функциональной спецификации* продукта, требования к продукту являются основным источником, определяющим, какие тесты будут разработаны. Для каждого требования пишется один или более тестов, которые в совокупности должны проверить выполнение данного требования в продукте.

##### Пример использования спецификации требований для разработки тестов.

Пусть задан следующий фрагмент набора требований для модели обмена транзакциями:

1. Функция DoTransaction должна принимать адрес и данные в соответствии с параметрами, создавать в очереди новый элемент, заполнять его адресную часть и часть полей данных переданной информацией и инициировать транзакцию
2. Функция DoAddressTenure должна принимать адрес в соответствии с параметрами, создавать в очереди новый элемент и заполнять его адресную часть
3. Функция DoDataTenure должна принимать данные в соответствии с параметрами, находить в очереди первый элемент с частично незаполненными полями данных, дополнять его переданной информацией и инициировать транзакцию

Концептуальное описание набора тестов, проверяющего спецификацию, может выглядеть следующим образом:

1. Вызвать DoTransaction с адресом и данными. Проверить появление в очереди еще одного элемента. Проверить появление на шине транзакции с правильными адресом и данными.
2. Вызвать DoAddressTenure с адресом. Проверить появление в очереди еще одного элемента. Проверить отсутствие новой транзакции на шине.
3. Вызвать DoDataTenure с данными. Проверить заполнение полей данных. Проверить появление на шине транзакции с правильными адресом и данными

#### Тестирование сценариев

Разработка тестов, основанных на использовании сценариев, осуществляется по следующей методике:

1. Определяется модель использования, включающая операционное окружение продукта и "актеров". Актером может быть пользователь, другой продукт, аппаратная часть и тому подобное, то есть все, с чем продукт обменивается информацией. Разделение на окружение и актеров условно и служит для описания оптимальных способов использования продукта.
2. Разрабатываются сценарии использования продукта. Описание сценария в зависимости от продукта и выбранного подхода может быть строго определенным, параметризованным или разрешать некоторую степень неопределенности. Например, описание сценария на языке MSC допускает задание параметризованных сценариев с возможностью переупорядочивания событий.
3. Разрабатывается набор тестов, покрывающих заданные сценарии. С учетом степени неопределенности, заложенной в сценарии, каждый тест может покрывать один сценарий, несколько сценариев, или, наоборот, часть сценария.

Использование сценариев не требует наличия полной *формальной спецификации* требований, но зато может потребовать больше времени на разработку и анализ.

Еще одна особенность *тестирования сценариев* заключается в том, что этот метод направляет тестирование на проверку конкретных режимов использования продукта, что позволяет находить дефекты, которые метод тестирования по требованиям может пропустить.

##### Пример использования спецификации требований для разработки тестов.

Так, для рассмотренного выше примера возможно создание следующего сценария и тестов.

1. Сценарий: пользователь имеет две независимые нити управления, одна из которых отвечает за генерацию полных транзакций посредством DoTransaction, а другая – за сбор транзакций из адресной части и части данных, когда эта информация приходит из разных источников. Таким образом, вторая нитка использует вызовы к DoAddressTenure и DoDataTenure.
2. Описание тестов: Вызвать DoAddressTenure c адресом А1, вызвать DoTransaction с адресом А2 и данными D2, вызвать DoDataTenure с данными D1. Проверить последовательное появление на шине двух транзакций: {А1, D1} и {А2, D2}

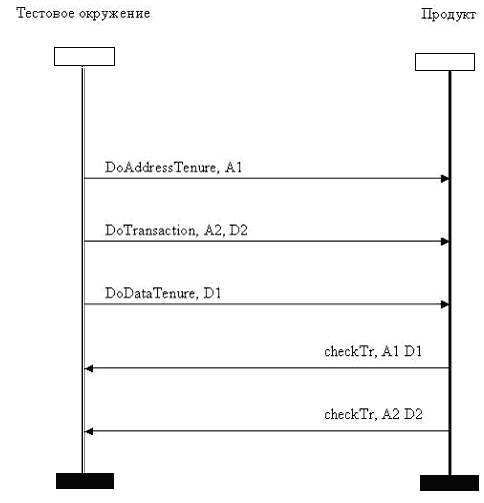
При выполнении этого теста было, в частности, обнаружено, что функция DoTransaction была реализована через вызовы к DoAddressTenure и DoDataTenure, что приводило к появлению на шине транзакций вида {А1, D2} и {А2, D1}. Подобный дефект может быть обнаружен с большим трудом, если разрабатывать тесты, основываясь только на спецификации требований.

#### Ручная разработка тестов

Наиболее распространенным способом разработки тестов является создание тестового кода вручную. Это наиболее гибкий способ разработки тестов, однако характерная для него производительность труда инженеров-тестировщиков в создании тестового кода не намного выше скорости создания кода продукта, а объемы тестового кода на практике зачастую превышают объем кода продукта в 10 раз. Учитывая этот факт, в современной индустрии все больше склоняются к более интеллектуальным способам получения тестового кода, таким как использование специальных тестовых *языков (скриптов*) и генерации тестов.

#### Генерация тестов

В настоящее время некоторые языки спецификаций, используемые для описания алгоритмов тестирования, могут быть использованы для генерации тестового кода. Рассмотрим генерацию кода из языка MSC. Тест, описанный выше, формализован на языке MSC ([Рис. 9.3](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=4#image.9.3)). Здесь каждая стрелка с пометкой DoTransaction, DoAddressTenure или DoDataTenure представлет собой вызов соответствующей функции продукта с передачей параметров. Стрелка checkTr соответствует проверке прохождения по шине транзакции с соответствующими параметрами. Каждая из стрелок диаграммы генератором тестов преобразуется в исполнимый код, при этом стрелкам, представляющим собой вызовы функций может соответствовать достаточно простой и маленький участок кода, вызывающий соответствующую функцию и проверяющий ее выходное значение на наличие ошибок.

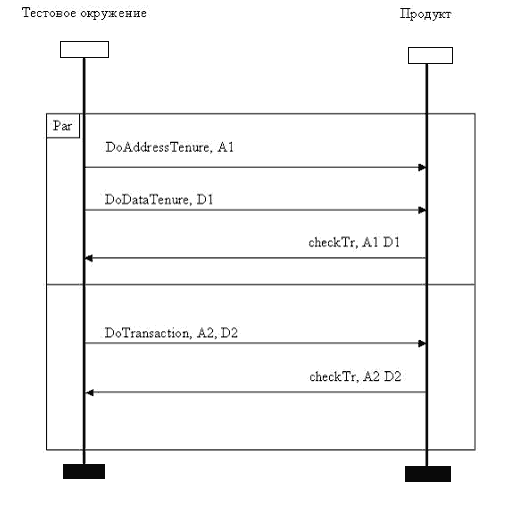


**Рис. 9.3.**Формальная запись сценарного теста на MSC

Следует отметить, что стрелки, соответствующие проверке транзакций, могут после генерации преобразоваться в достаточно сложный код, который будет выполнять ожидание появления транзакции на шине в течение заданного при генерации времени - тайм-аута, проверять фазы транзакции и сверять вычисленные значения параметров с заданными эталонными значениями.

В результате в рассматриваемом примере выигрыш от применения генерационного подхода достигается в основном за счет использования наглядного визуального представления тестов, что может быть нивелировано затратами на создание генерационного сценария на MSC.

Возможна куда более эффективная формализация MSC сценария для генерации тестов. [Рис. 9.4](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=4#image.9.4) представляет другой способ формализации для теста, выполняющего те же самые проверки.



**Рис. 9.4.**Формальная запись сценарного теста на MSC с использованием параллелизма.

В MSC на [Рис. 9.4](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=4#image.9.4) проверки транзакций сгруппированы с порождающими их вызовами в отдельные фрагменты, а параллелизм, используемый при исполнении фрагментов, задан через Par – формальную конструкцию, применяемую для изображения параллелизма в языке MSC. При генерации тестов по диаграмме [Рис. 9.4](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=4#image.9.4) тестовый генератор перебирает все возможные и неповторяющиеся варианты вызова тестируемых функций, сохраняя при этом корректность порядка проверок, что в данном примере дает три сгенерированных теста. Несложно видеть, что затраты на создание диаграммы [Рис. 9.4](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=4#image.9.4) не сильно отличаются от затрат на диаграмму [Рис. 9.3](https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=4#image.9.3), в то время как количество тестов увеличивается в три раза.

Таким образом, использование методики генерации тестового кода по формализованным MSC диаграммам позволяет значительно поднять производительность тестирования, а также преобразовать формализацию (кодировку) сценариев в достаточно интеллектуальную деятельность.

### **Выполнение тестов**

Рассмотрим два основных подхода к выполнению тестов: подход *ручного тестирования* и подход автоматического исполнения (*прогон) тестов*. Подходы рассмотрены на примере тестирования продукта, поддерживающего *интерфейс командной строки*. Тесты описывают вызов продукта с параметрами и проверку возвращаемого значения в виде фиксируемых при прогоне – текста из STDOUT и состояния некоторых файлов, зависящего от входных параметров.

#### Ручное тестирование

*Ручное тестирование* заключается в выполнении задокументированной процедуры, где описана методика выполнения тестов, задающая порядок тестов и для каждого теста - список значений параметров, который подается на вход, и список результатов, ожидаемых на выходе. Поскольку процедура предназначена для выполнения человеком, в ее описании для краткости могут использоваться некоторые значения по умолчанию, ориентированные на здравый смысл, или ссылки на информацию, хранящуюся в другом документе.

##### Пример фрагмента процедуры

1. Подать на вход три разных целых числа.
2. Запустить тестовое исполнение.
3. Проверить, соответствует ли полученный результат таблице [ссылка на документ1] с учетом поправок [ссылка на документ2].
4. Убедиться в понятности и корректности выдаваемой сопроводительной информации.

В приведенной процедуре тестировщик использует два дополнительных документа, а также собственное понимание того, какую сопроводительную информацию считать "понятной и корректной". Успех от использования процедурного подхода достигается в случае однозначного понимания тестировщиком всех пунктов процедуры. Например, в п.1 приведенной процедуры не уточняется, из какого диапазона должны быть заданы три целых числа, и не описывается дополнительно, какие числа считаются "разными".

#### Автоматизированное тестирование

Попытка автоматизировать приведенный выше тест приводит к созданию скрипта, задающего тестируемому продукту три конкретных числа и перенаправляющего вывод продукта в файл с целью его анализа, а также содержащего конкретное значение желаемого результата, с которым сверяется получаемое при прогоне теста значение. Таким образом, вся необходимая информация должна быть явно помещена в текст (скрипт) теста, что требует дополнительных по сравнению с ручным подходом усилий. Также дополнительных усилий и времени требует создание разборщика вывода (программы согласования форматов представления эталонных значений из теста и вычисляемых при прогоне результатов) и, возможно, создание базы хранения состояний эталонных данных.

##### Пример скрипта

Приведем пример последовательности действий, закладываемых в скрипт:

1. Выдать на консоль имя или номер теста и время его начала.
2. Вызвать продукт с фиксированными параметрами.
3. Перенаправить вывод продукта в файл.
4. Проверить возвращенное продуктом значение. Оно должно быть равно ожидаемому (эталонному) результату, зафиксированному в тесте.
5. Проверить вывод продукта, сохраненный в файле (п.3), на равенство заранее приготовленному эталону.
6. Выдать на консоль результаты теста в виде вердикта PASS/FAIL и в случае FAIL - краткого пояснения, какая именно проверка не прошла.
7. Выдать на консоль время окончания теста.

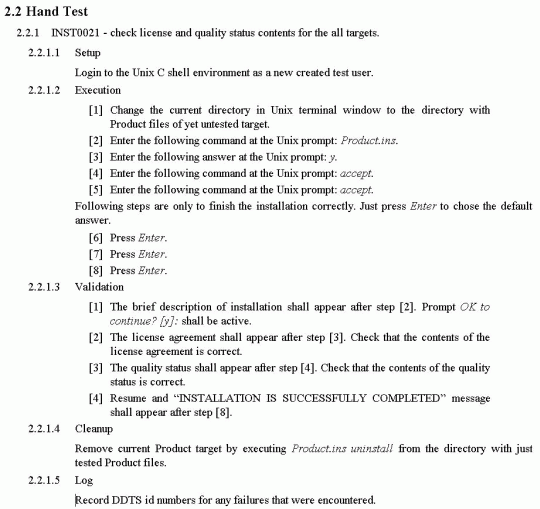
#### Сравнение ручного и автоматизированного тестирования

Результаты сравнения приведены в [Табл. 10.1](https://www.intuit.ru/studies/courses/48/48/lecture/1442?page=1#table.10.1). Сравнение показывает тенденцию современного тестирования, ориентирующую на максимальную автоматизацию процесса тестирования и генерацию тестового кода, что позволяет справляться с большими объемами данных и тестов, необходимых для обеспечения качества при производстве программных продуктов.

|  |  |  |
| --- | --- | --- |
| Таблица 10.1. Сравнение ручного и автоматизированного подхода | | |
|  | **Ручное** | **Автоматизированное** |
| **Задание входных значений** | Гибкость в задании данных. Позволяет использовать разные значения на разных циклах прогона тестов, расширяя покрытие | Входные значения строго заданы |
| **Проверка результата** | Гибкая, позволяет тестировщику оценивать нечетко сформулированные критерии | Строгая. Нечетко сформулированные критерии могут быть проверены только путем сравнения с эталоном |
| **Повторяемость** | Низкая. Человеческий фактор и нечеткое определение данных приводят к неповторяемости тестирования | Высокая |
| **Надежность** | Низкая. Длительные *тестовые циклы* приводят к снижению внимания тестировщика | Высокая, не зависит от длины *тестового цикла* |
| **Чувствительность к незначительным изменениям в продукте** | Зависит от детальности описания процедуры. Обычно тестировщик в состоянии выполнить тест, если внешний вид продукта и текст сообщений несколько изменились | Высокая. Незначительные изменения в интерфейсе часто ведут к коррекции эталонов |
| **Скорость выполнения *тестового набора*** | Низкая | Высокая |
| **Возможность генерации тестов** | Отсутствует. Низкая скорость выполнения обычно не позволяет исполнить сгенерированный набор тестов | Поддерживается |

#### Тестовые процедуры

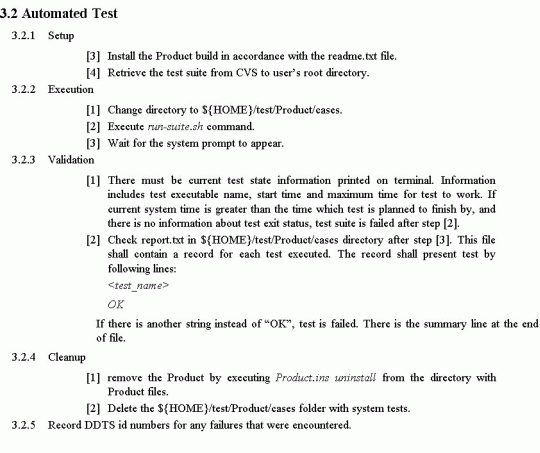
***Тестовые процедуры*** - это формальный документ, содержащий описание необходимых шагов для выполнения *тестового набора* . В случае ручных тестов *тестовые процедуры* содержат полное описание всех шагов и проверок, позволяющих протестировать продукт и вынести вердикт PASS/FAIL.



**Рис. 10.1.**Пример фрагмента тестовой процедуры для ручного тестирования

Процедуры должны быть составлены таким образом, чтобы любой инженер, не связанный с данным проектом, был способен адекватно провести цикл тестирования, обладая только самыми базовыми знаниями о применяющемся инструментарии. Пример фрагмента *тестовой процедуры* для *ручного тестирования* приведен на [Рис. 10.1](https://www.intuit.ru/studies/courses/48/48/lecture/1442?page=2#image.10.1)

В случае описания автоматизированных тестов *тестовые процедуры* должны содержать достаточную информацию для запуска тестов и анализа результатов. Пример фрагмента такой процедуры приведен на [Рис. 10.2](https://www.intuit.ru/studies/courses/48/48/lecture/1442?page=2#image.10.2)



**Рис. 10.2.**Пример фрагмента автоматизированной тестовой процедуры

#### Описание тестов

Описание тестов разрабатывается для облегчения анализа и поддержки *тестового набора*. Описание может быть реализовано в произвольной форме, но при этом должны выполнять следующие задачи:

1. Анализировать степень покрытия продукта тестами на основании описания *тестового набора*.
2. Для любой функции тестируемого продукта найти тесты, в которых функция используется.
3. Для любого теста определить все функции и их сочетания, которые данный тест использует (затрагивает).
4. Понять структуру и взаимосвязи тестовых файлов.
5. Понять принцип построения системы *автоматизации тестирования*.

#### Документирование и жизненный цикл дефекта

Каждый дефект, обнаруженный в процессе тестирования, должен быть задокументирован и отслежен. При обнаружении нового дефекта его заносят в базу дефектов. Для этого лучше всего использовать специализированные базы, поддерживающие хранение и отслеживание дефектов - типа DDTS . При занесении нового дефекта рекомендуется указывать, как минимум, следующую информацию:

1. Наименование подсистемы, в которой обнаружен дефект.
2. Версия продукта (номер build ), на котором дефект был найден.
3. Описание дефекта.
4. Описание процедуры (шагов, необходимых для воспроизведения дефекта).
5. Номер теста, на котором дефект был обнаружен.
6. Уровень дефекта, то есть степень его серьезности с точки зрения критериев качества продукта или заказчика.

Занесенный в базу дефектов новый дефект находится в состоянии " **New** " . После того, как команда разработчиков проанализирует дефект, он переводится в состояние " **Open** " с указанием конкретного разработчика, ответственного за исправление дефекта. После исправления дефект переводится разработчиком в состояние " **Resolved** ". При этом разработчик должен указать следующую информацию:

1. Причину возникновения дефекта.
2. Место исправления, как минимум, с точностью до исправленного файла.
3. Краткое описание того, что было исправлено.
4. Время, затраченное на исправление.

После этого тестировщик проверяет, действительно ли дефект был исправлен и если это так, переводит его в состояние " **Verified** ". Если тестировщик не подтвердит факт исправления дефекта, то состояние дефекта изменяется снова на " **Open** ".

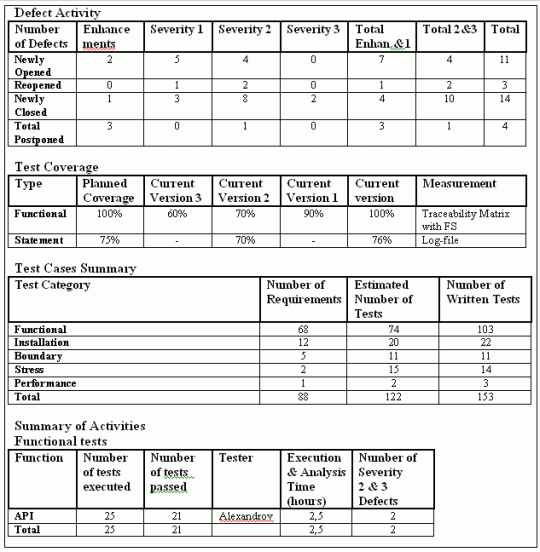
Если проектная команда принимает решение о том, что некоторый дефект исправляться не будет, то такой дефект переводится в состояние " **Postponed** " с указанием лиц, ответственных за это решение, и причин его принятия.

#### Тестовый отчет

*Тестовый отчет* обновляется после каждого цикла тестирования и должен содержать следующую информацию для каждого цикла:

1. Перечень функциональности в соответствии с пунктами требований, запланированный для тестирования на данном цикле, и реальные данные по нему.
2. Количество выполненных тестов – запланированное и реально исполненное.
3. Время, затраченное на тестирование каждой функции, и общее время тестирования.
4. Количество найденных дефектов.
5. Количество повторно открытых дефектов.
6. Отклонения от запланированной последовательности действий, если таковые имели место.
7. Выводы о необходимых корректировках в системе тестов, которые должны быть сделаны до следующего *тестового цикла*.

Пример фрагмента из *тестового отчета* представлен на [Рис. 10.3](https://www.intuit.ru/studies/courses/48/48/lecture/1442?page=2#image.10.3). Приведенный фрагмент отчета содержит примерные данные для четырех циклов тестирования и иллюстрирует структуру отчета. Такой вид отчет имеет после тестирования, перед началом цикла тестирования поля не заполнены, заполнение осуществляется по окончании соответствующего цикла.



**Рис. 10.3.**Фрагмент тестового отчета

### **Оценка качества тестов**

Тесты нуждаются в контроле качества так же, как и тестируемый продукт. Поскольку тесты для продукта являются своего рода эталоном его структурных и поведенческих характеристик, закономерен вопрос о том, насколько адекватен эталон. Для оценки качества тестов используются различные методы, наиболее популярные из которых кратко рассмотрены ниже.

#### Тестовые метрики

Существует устоявшийся набор тестовых метрик, который помогает определить эффективность тестирования и текущее состояние продукта. К таким метрикам относятся следующие:

1. Покрытие функциональных требований.
2. Покрытие кода продукта. Наиболее применимо для модульного уровня тестирования.
3. Покрытие множества сценариев.
4. Количество или плотность найденных дефектов. Текущее количество дефектов сравнивается со средним для данного типа продуктов с целью установить, находится ли оно в пределах допустимого статистического отклонения. При этом обнаруженные отклонения как в большую, так и в меньшую сторону приводят к анализу причин их появления и, если необходимо, к выработке корректирующих действий.
5. Соотношение количества найденных дефектов с количеством тестов на данную функцию продукта. Сильное расхождение этих двух величин говорит либо о неэффективности тестов (когда большое количество тестов находит мало дефектов) либо о плохом *качестве данного* участка кода (когда найдено большое количество дефектов на не очень большом количестве тестов).
6. Количество найденных дефектов, соотнесенное по времени, или скорость поиска дефектов. Если производная такой функции близка к нулю, то продукт обладает качеством, достаточным для окончания тестирования и поставки заказчику.

#### Обзоры тестов и стратегии

Тестовый код и стратегия тестирования, зафиксированные в виде документов, заметно улучшаются, если подвергаются коллективному обсуждению. Такие обсуждения называются обзорами ( **review** ). Существует принятая в организации процедура проведения и оценки результатов обзора. Обзоры наряду с тестированием образуют мощный набор методов борьбы с ошибками с целью повышения качества продукта. Цели *обзоров тестовой стратегии* и тестового кода различны.

Цели *обзора тестовой стратегии*:

1. Установить достаточность проверок, обеспечиваемых тестированием.
2. Проанализировать оптимальность покрытия или адекватность распределения количества планируемых тестов по функциональности продукта.
3. Проанализировать оптимальность подхода к разработке кода, генерации кода, *автоматизации тестирования*.

Цели обзора тестового кода:

1. Установить соответствие *тестового набора* тестовой стратегии.
2. Проверить правильность кодирования тестов.
3. Оценить достигнутую степень качества кода, исходя из требований по стандартам, простоте поддержки, наличию комментариев и т.п.
4. Если необходимо, проанализировать оптимальность тестового кода с целью удовлетворения требований к быстродействию и объему.