***Слайд 1***

**Лекция 5**

# Тестирование программного обеспечения

***Слайд 2***

**Учебные вопросы:**

**1. Тестирование программного обеспечения, основные понятия.**

**2. Виды тестирования ПО.**

**3. Тестовые артефакты, тест-дизайн при проведении тестирования.**

***Слайд 3***

## 1. Тестирование программного обеспечения, основные понятия

*Важным этапом жизненного цикла ПО, определяющим качество и надёжность системы, является* ***тестирование****.*

*Тестирование* - процесс выполнения программ с намерением найти ошибки и включает в себя следующие этапы:

* автономное тестирование;
* тестирование сопряжений;
* тестирование функций;
* комплексное тестирование;
* тестирование полноты и корректности документации;
* тестирование конфигураций.

*Надежность ПО повышается также с помощью применения различных* ***методов тестирования****. Полное тестирование ПО невозможно.* Обычно применяют следующие виды тестирования:

* тестирование ветвей;
* математическое доказательство правильности алгоритма решения задачи (верификация);
* символическое тестирование (статистическое);
* динамическое тестирование;
* тестирование путей выполнения программы;
* функциональное тестирование;
* проверки по времени выполнения программы;
* проверка по использованию ресурсов;
* стрессовое тестирование.

***Слайд 4***

**Основные понятия**

**Тестирование программного обеспечения** **(Software Testing)** - проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

**Верификация (Verification)** - это процесс оценки системы или её компонентов с целью определения удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа.

**Валидация (Validation)** - это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

**План тестирования** - это документ, описывающий весь объем работ по тестированию, *начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.*

**Тест дизайн** - это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы), в соответствии с определёнными ранее критериями качества и целями тестирования.

**Тестовый случай** (Test Case) - это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

***Баг репорт*** *- это документ, описывающий ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.*

**Тестовое покрытие** - это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.

***Детализация Тест Кейсов*** *(****Test Case Specification****) - это уровень детализации описания тестовых шагов и требуемого результата, при котором обеспечивается разумное соотношение времени прохождения к тестовому покрытию.*

***Слайд 5***

## 2. Виды тестирования

Все **виды тестирования программного обеспечения**, в зависимости от преследуемых целей, можно условно разделить на следующие группы:

1. ***Функциональные*** - *базируются на функциях и особенностях, а также взаимодействии с другими системами, и могут быть представлены на всех уровнях тестирования (компонентном или модульном, интеграционном, системном и приемочном)*. ***Функциональные виды тестирования рассматривают внешнее поведение системы.***
2. ***Нефункциональные*** - *описывают тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами.* ***В целом, это тестирование того, "Как" система работает.*** *К нефункциональным относятся все виды тестирования производительности (нагрузочное тестирование, стрессовое тестирование, тестирование стабильности или надежности, объемное тестирование), тестирование установки, тестирование удобства пользования, тестирование на отказ и восстановление, конфигурационное тестирование.*
3. ***Связанные с изменениями*** *виды тестирования описывают тесты, которые должны быть проведены после изменений (например, исправление бага). Следующие виды тестирования проводятся* ***после установки программного обеспечения для подтверждения работоспособности приложения или правильности осуществленного исправления дефекта****: дымовое тестирование (Smoke testing), регрессионное тестирование (Regression testing), Тестирование сборки (Build verification test), санитарное тестирование или проверка согласованности (Sanity testing).*

***Слайд 6***

### 2.1. Функциональное тестирование

**Функциональное тестирование** рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом.

*Как правило, эти функции описываются в требованиях, функциональных спецификациях или в виде случаев использования системы (****use cases****).*

Тестирование функциональности может проводиться в двух аспектах:

* требования
* бизнес-процессы

Тестирование в перспективе «требования» использует спецификацию функциональных требований к системе как основу для дизайна тестовых случаев. *В этом случае* ***необходимо*** *сделать* ***список того, что будет тестироваться****, а что нет,* ***расставить приоритеты требованиям на основе рисков***, а **на основе этого актуализировать тестовые сценарии** (test cases). *Это позволит сфокусироваться и не упустить при тестировании наиболее важный функционал.*

Тестирование в перспективе «бизнес-процессы» использует знание этих самых бизнес-процессов, которые описывают сценарии ежедневного использования системы. *В этой перспективе тестовые сценарии (****test scripts****), как правило, основываются на случаях использования системы (use cases).*

**Преимущества функционального тестирования**:

* имитирует фактическое использование системы;

**Недостатки функционального тестирования**:

* возможность упущения логических ошибок в программном обеспечении;
* вероятность избыточного тестирования.

***Слайд 7***

#### 2.1.1 Компонентное (модульное) тестирование

**Компонентное (модульное) тестирование проверяет функциональность и ищет дефекты в частях приложения**, которые доступны и могут быть протестированы по-отдельности (**модули программ, объекты, классы, функции и т.д.**). Обычно компонентное (модульное) тестирование проводится вызывая код, который необходимо проверить и при поддержке сред разработки, *таких как фреймворки (frameworks - каркасы) для модульного тестирования или инструменты для отладки*, проверяется на наличие дефектов. Все найденные дефекты, как правило, исправляются в коде без формального их описания в системе менеджмента багов (Bug Tracking System).

*Один из наиболее эффективных подходов к компонентному (модульному) тестированию - это* ***подготовка автоматизированных тестов*** *до начала основного кодирования (разработки) программного обеспечения. Это называется разработка от тестирования (****test-driven development****) или подход тестирования вначале (****test first approach****). При этом подходе создаются и интегрируются небольшие куски кода, напротив которых запускаются тесты, написанные до начала кодирования. Разработка ведется до тех пор, пока все тесты не будут успешно пройдены.*

***Слайд 8***

#### 2.1.2 Интеграционное тестирование

Интеграционное тестирование предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).

**Уровни интеграционного тестирования:**

* **Компонентный интеграционный уровень** (*Component Integration testing*)

Проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.

* **Системный интеграционный уровень** (*System Integration Testing*)

Проверяется взаимодействие между разными системами после проведения системного тестирования.

**Подходы к интеграционному тестированию:**

* **Снизу вверх (восходящее тестирование)** (*Bottom Up Integration*)

Все низкоуровневые модули, процедуры или функции собираются воедино и затем тестируются. *После чего собирается следующий уровень модулей для проведения интеграционного тестирования. Данный подход считается полезным, если все или практически все модули, разрабатываемого уровня, готовы. Также данный подход помогает определить по результатам тестирования уровень готовности приложения.*

* **Сверху вниз (нисходящее тестирование)** (*Top Down Integration*)

Вначале тестируются все высокоуровневые модули, и постепенно один за другим добавляются низкоуровневые. *Все модули более низкого уровня симулируются заглушками с аналогичной функциональностью, затем по мере готовности они заменяются реальными активными компонентами. Таким образом мы проводим тестирование сверху вниз.*

* **Большой взрыв** (*"Big Bang" Integration*)

Все или практически все разработанные модули собираются вместе в виде законченной системы или ее основной части, и затем проводится интеграционное тестирование. Такой подход очень хорош для сохранения времени. Однако если тест кейсы и их результаты записаны не верно, то сам процесс интеграции сильно осложнится, что станет преградой для команды тестирования при достижении основной цели интеграционного тестирования

***Слайд 9***

#### 2.1.3 Системное тестирование

Основной задачей системного тестирования является проверка как функциональных так и не функциональных требований в системе в целом. *При этом* ***выявляются дефекты****, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д. Для минимизации рисков, связанных с особенностями поведения в системы в той или иной среде, во время тестирования рекомендуется использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи.*

Можно выделить два подхода к системному тестированию:

* **на базе требований** (*requirements based*)

Для каждого требования пишутся тестовые случаи, проверяющие выполнение данного требования.

* **на базе случаев использования** (*use case based*)

На основе представления о способах использования продукта создаются случаи использования системы (**Use Cases**). *По конкретному случаю использования можно определить один или более сценариев. На проверку каждого сценария пишутся тестовые случаи, которые должны быть протестированы.*

#### 2.1.4 Приемочное тестирование

Формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью:

* определения удовлетворяет ли система приемочным критериям;
* вынесения решения заказчиком или другим уполномоченным лицом принимается приложение или нет.

*Приемочное тестирование выполняется на основании набора типичных тестовых случаев и сценариев, разработанных на основании требований к данному приложению.*

**Решение о проведении приемочного тестирования** принимается, когда:

* продукт достиг необходимого уровня качества;
* заказчик ознакомлен с **Планом Приемочных Работ** или иным документом, где описан набор действий, связанных с проведением приемочного тестирования, *дата проведения, ответственные и т.д.*

***Фаза приемочного тестирования*** *длится до тех пор, пока заказчик не выносит решение об отправлении приложения на доработку или выдаче приложения.*

***Слайд 10***

### 2.2 Нефункциональное тестирование

#### 2.2.1 Тестирование взаимодействия

*С развитием сетевых технологий и интернета взаимодействие разных систем, сервисов и приложений друг с другом приобрело значительную актуальность, так как любые связанные с этим проблемы могут привести к падению авторитета компании, что как следствие повлечет за собой финансовые потери.* **Тестирование взаимодействия** (**Interoperability Testing**) – это тестирование, проверяющее способность приложения взаимодействовать с одним и более компонентами или системами и включающее в себя тестирование совместимости и интеграционное тестирование. *Программное обеспечение с хорошими характеристиками взаимодействия может быть легко интегрировано с другими системами, не требуя каких-либо серьезных модификаций. В этом случае, количество изменений и время, требуемое на их выполнение, могут быть использованы для измерения* ***возможности взаимодействия.***

#### 2.2.2 Тестирование безопасности

**Тестирование безопасности** - это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

Общая стратегия безопасности основывается на трех основных принципах:

1. **конфиденциальность** - это сокрытие определенных ресурсов или информации. Под конфиденциальностью можно понимать ограничение доступа к ресурсу некоторой категории пользователей, или другими словами, при каких условиях пользователь авторизован получить доступ к данному ресурсу.
2. **целостность** основана на двух критериях:
   1. **доверие**: ожидается, что ресурс будет изменен только соответствующим способом определенной группой пользователей
   2. **повреждение и восстановление**: в случае когда данные повреждаются или неправильно меняются авторизованным или не авторизованным пользователем, вы должны определить на сколько важной является процедура восстановления данных.
3. **доступность** представляет собой требования о том, что ресурсы должны быть доступны авторизованному пользователю, внутреннему объекту или устройству. *Как правило, чем более критичен ресурс, тем выше уровень доступности должен быть.*

***Слайд 11***

Рассмотрим наиболее распространенные **виды уязвимостей**:

* **XSS (Cross-Site Scripting)** - это вид уязвимости программного обеспечения (Web приложений), при которой, на генерированной сервером странице, выполняются вредоносные скрипты, с целью атаки клиента. *Сами по себе XSS атаки могут быть очень разнообразными. Злоумышленники могут попытаться украсть cookie-файлы, перенаправить вас на сайт, где произойдет более серьезная атака, загрузить в память какой-либо вредоносный объект и т.д., лишь* ***разместив вредоносный скрипт*** *у вас на сайте.*
* **XSRF / CSRF (Request Forgery) (Межсайтовая подделка запроса)** - это вид уязвимости, позволяющий использовать недостатки HTTP протокола, при этом злоумышленники работают по следующей схеме: ссылка на вредоносный сайт устанавливается на странице, пользующейся доверием у пользователя, при переходе по вредоносной ссылке выполняется скрипт, сохраняющий личные данные пользователя (пароли, платежные данные и т.д.), либо отправляющий СПАМ сообщения от лица пользователя, либо изменяет доступ к учетной записи пользователя, для получения полного контроля над ней.

*Наиболее частыми CSRF атаками являются атаки, использующие HTML <IMG> тэг или Javascript объект image. Чаще всего атакующий добавляет необходимый код в электронное письмо или выкладывает на веб-сайт, таким образом, что при загрузке страницы осуществляется запрос, выполняющий вредоносный код.*

***Слайд 12***

* **Code injections (SQL, PHP, ASP и т.д.) (процесс инъекции своего кода в память чужого приложения с дальнейшим его выполнением)** - это вид уязвимости, при котором становится возможным осуществить запуск исполняемого кода с целью получения доступа к системным ресурсам, несанкционированного доступа к данным либо выведения системы из строя.
* **Server-Side Includes (SSI) Injection** - это вид уязвимости, использующий вставку серверных команд в HTML код или запуск их напрямую с сервера.

*В зависимости от типа операционной системы команды могут быть разными.*

* **Authorization Bypass** - это вид уязвимости, при котором возможно получить несанкционированный доступ к учетной записи или документам другого пользователя

*Пользователь А может получить доступ к документам пользователя Б. Допустим, есть реализация, где при просмотре своего профиля, содержащего конфиденциальную информацию, в URL страницы передается userID, в данном случае есть смысл попробовать подставить вместо своего userID номер другого пользователя. И если данные пользователя доступны, значит дефект(уязвимость) обнаружена.*

***Слайд 13***

#### 2.2.3 Тестирование производительности

##### **2.2.3.1 Нагрузочное тестирование**

**Нагрузочное тестирование** – это автоматизированное тестирование, имитирующее работу определенного количества бизнес пользователей на каком-либо общем (разделяемом ими) ресурсе.

Задачами нагрузочного тестирования является определение масштабируемости приложения под нагрузкой, при этом происходит:

* + измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций;
  + определение количества пользователей, одновременно работающих с приложением;
  + определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций);
  + исследование производительности на высоких, предельных, стрессовых нагрузках.

***Слайд 14***

##### **2.2.3.2 Стрессовое тестирование**

**Стрессовое тестирование** (**Stress Testing**) позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса и также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса***. Стрессом в данном контексте может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера.*** *Также одной из задач при стрессовом тестировании может быть оценка деградации производительности, таким образом, цели стрессового тестирования могут пересекаться с целями тестирования производительности.*

##### **2.2.3.3 Объемное тестирование**

**Объемное тестирование** (**Volume Testing**) - получение оценки производительности при увеличении объемов данных в базе данных приложения, при этом происходит:

1. измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций

2. может производиться определение количества пользователей, одновременно работающих с приложением.

##### **2.2.3.4 Тестирование стабильности или надежности**

**Тестирование стабильности или надежности.** Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Время выполнения операций может играть в данном виде тестирования второстепенную роль. При этом на первое место выходит отсутствие утечек памяти, перезапусков серверов под нагрузкой и другие аспекты влияющие именно на стабильность работы.

***Слайд 15***

#### 2.2.4 Тестирование установки

Тестирование установки направленно на проверку успешной инсталляции и настройки, а также обновления или удаления программного обеспечения. *В настоящий момент наиболее распространена установка ПО при помощи* ***инсталляторов.***

*В реальных условиях инсталляторов может* ***не быть****. В этом случае придется* ***самостоятельно выполнять установку программного обеспечения****, используя документацию в виде инструкций или readme файлов, шаг за шагом описывающих все необходимые действия и проверки.*

В распределенных системах, где приложение разворачивается на уже работающем окружении, простого набора инструкций может быть мало. Для этого, зачастую, пишется план установки, включающий не только шаги по инсталляции приложения, но и шаги отката к предыдущей версии, в случае неудачи. Сам по себе план установки также должен пройти процедуру тестирования для избежание проблем при выдаче в реальную эксплуатацию. *Особенно это актуально, если установка выполняется на системы, где каждая минута простоя - это потеря репутации и большого количества средств, например: банки, финансовые компании или даже баннерные сети. Поэтому тестирование установки можно назвать одной из важнейших задач по обеспечению качества программного обеспечения.*

*Именно такой комплексный подход с написанием планов, пошаговой проверкой установки и отката инсталляции, полноправно можно назвать тестированием установки или Installation Testing.*

#### 2.2.5 Конфигурационное тестирование

**Конфигурационное тестирование** (**Configuration Testing**) или **тестирование портируемости** — специальный вид тестирования, направленный на проверку работы программного обеспечения при различных конфигурациях системы (заявленных платформах, поддерживаемых драйверах, при различных конфигурациях компьютеров и т.д.)

В зависимости от типа проекта конфигурационное тестирование может иметь разные цели:

1. Проект по профилированию работы системы  
   **Цель Тестирования**: определить оптимальную конфигурацию оборудования, обеспечивающую требуемые характеристики производительности и времени реакции тестируемой системы.
2. Проект по миграции системы с одной платформы на другую  
   **Цель Тестирования**: Проверить объект тестирования на совместимость с объявленным в спецификации оборудованием, операционными системами и программными продуктами третьих фирм.

***Слайд 16***

Для клиент-серверных приложений **конфигурационное тестирование** можно условно разделить на два уровня (для некоторых типов приложений может быть актуален только один):

1. Серверный
2. Клиентский

На первом (серверном) уровне, тестируется взаимодействие выпускаемого программного обеспечения с окружением, в которое оно будет установлено:

1. Аппаратные средства (тип и количество процессоров, объем памяти, характеристики сети / сетевых адаптеров и т.д.)
2. Программные средства (ОС, драйвера и библиотеки, стороннее ПО, влияющее на работу приложения и т.д.)

*Основной упор здесь делается на тестирование с целью определения оптимальной конфигурации оборудования, удовлетворяющего требуемым характеристикам качества (эффективность, портативность, удобство сопровождения, надежность).*

На следующем (клиентском) уровне, программное обеспечение тестируется с позиции его конечного пользователя и конфигурации его рабочей станции. На этом этапе будут протестированы следующие характеристики: удобство использования, функциональность. Для этого необходимо будет провести ряд тестов с различными конфигурациями рабочих станций:

1. Тип, версия и битность операционной системы (подобный вид тестирования называется **кросс-платформенное тестирование**)
2. Тип и версия Web браузера, *в случае если тестируется Web приложение* (подобный вид тестирования называется **кросс-браузерное тестирование**)
3. Тип и модель видео адаптера (при тестировании игр это очень важно)
4. Работа приложения при различных разрешениях экрана
5. Версии драйверов, библиотек и т.д. (для JAVA приложений версия JAVA машины очень важна, тоже можно сказать и для .NET приложений касательно версии .NET библиотеки) и т.д.

Перед началом проведения конфигурационного тестирования рекомендуется:

* создавать матрицу покрытия *(****матрица покрытия*** *- это таблица, в которую заносят все возможные конфигурации)*,
* определять приоритетность конфигураций (на практике, скорее всего, все желаемые конфигурации проверить не получится),
* шаг за шагом, в соответствии с расставленными приоритетами, проверяют каждую конфигурацию.

***Слайд 17***

#### 2.2.6 Тестирование удобства пользования

*Для того чтобы приложение было популярным, ему мало быть функциональным – оно должно быть еще и удобным. Если задуматься, интуитивно понятные приложения экономят нервы пользователям и затраты работодателя на обучение. А значит они более конкурентоспособные!*

**Тестирование удобства пользования** - это метод тестирования, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий.

Тестирование удобства пользования дает оценку уровня удобства использования приложения по следующим пунктам:

* **производительность, эффективность** (**efficiency**) - сколько времени и шагов понадобится пользователю для завершения основных задач приложения, например, размещение новости, регистрации, покупка и т.д.? (*меньше - лучше*)
* **правильность** (**accuracy**) - сколько ошибок сделал пользователь во время работы с приложением? (*меньше - лучше*)
* **активизация в памяти** (**recall**) – как много пользователь помнит о работе приложения после приостановки работы с ним на длительный период времени? (*повторное выполнение операций после перерыва должно проходить быстрее, чем у нового пользователя*)
* **эмоциональная реакция** (**emotional response**) – как пользователь себя чувствует после завершения задачи - растерян, испытал стресс? Порекомендует ли пользователь систему своим друзьям? (*положительная реакция - лучше*)

***Слайд 18***

#### 2.2.7 Тестирование на отказ и восстановление

**Тестирование на отказ и восстановление** (**Failover and Recovery Testing**) проверяет тестируемый продукт с точки зрения способности противостоять и успешно восстанавливаться после возможных сбоев, возникших в связи с ошибками программного обеспечения, отказами оборудования или проблемами связи (например, отказ сети). **Целью** данного **вида тестирования** является проверка систем восстановления (или дублирующих основной функционал систем), которые, в случае возникновения сбоев, обеспечат сохранность и целостность данных тестируемого продукта.

***Тестирование на отказ и восстановление*** *очень важно для систем, работающих по принципу “24x7”.* *Если Вы создаете продукт, который будет работать, например, в интернете, то без проведения данного вида тестирования Вам просто не обойтись. Т.к. каждая минута простоя или потеря данных в случае отказа оборудования, может стоить вам денег, потери клиентов и репутации на рынке.*

**Методика** подобного **тестирования** заключается в симулировании различных условий сбоя и последующем изучении, и оценке реакции защитных систем. В процессе подобных проверок выясняется, была ли достигнута требуемая степень восстановления системы после возникновения сбоя.

*Для наглядности рассмотрим некоторые варианты подобного тестирования и общие методы их проведения. Объектом тестирования в большинстве случаев являются весьма вероятные эксплуатационные проблемы, такие как:*

* *Отказ электричества на компьютере-сервере*
* *Отказ электричества на компьютере-клиенте*
* *Незавершенные циклы обработки данных (прерывание работы фильтров данных, прерывание синхронизации).*
* *Объявление или внесение в массивы данных невозможных или ошибочных элементов.*
* *Отказ носителей данных.*

*Данные ситуации могут быть воспроизведены, как только достигнута некоторая точка в разработке, когда все системы восстановления или дублирования готовы выполнять свои функции. Технически реализовать тесты можно следующими путями:*

* *Симулировать внезапный отказ электричества на компьютере (обесточить компьютер).*
* *Симулировать потерю связи с сетью (выключить сетевой кабель, обесточить сетевое устройство)*
* *Симулировать отказ носителей (обесточить внешний носитель данных)*
* *Симулировать ситуацию наличия в системе неверных данных (специальный тестовый набор или база данных).*

***Слайд 19***

### 2.3 Тестирование изменений

#### 2.3.1 Дымовое тестирование

*Понятие дымового тестирования произошло из инженерной среды: "При вводе в эксплуатацию нового оборудования ("железа") считалось, что тестирование прошло удачно, если из установки не пошел дым".*

В области же программного обеспечения, ***дымовое тестирование*** рассматривается как короткий цикл тестов, выполняемый для подтверждения того, что после сборки кода (нового или исправленного) устанавливаемое приложение, стартует и выполняет основные функции.

*Вывод о работоспособности основных функций делается на основании результатов поверхностного тестирования наиболее важных модулей приложения на предмет возможности выполнения требуемых задач и наличия быстронаходимых критических и блокирующих дефектов. В случае отсутствия таковых дефектов дымовое тестирование объявляется пройденным, и приложение передается для проведения полного цикла тестирования, в противном случае, дымовое тестирование объявляется проваленным, и приложение уходит на доработку.*

*Для облегчения работы, экономии времени и людских ресурсов рекомендуется внедрить автоматизацию тестовых сценариев для дымового тестирования.*

#### 2.3.2 Тестирование сборки (Build Verification Test)

Тестирование, направленное на определение соответствия, выпущенной версии, критериям качества для начала тестирования. *По своим целям является аналогом дымового тестирования, направленного на приемку новой версии в дальнейшее тестирование или эксплуатацию. Вглубь оно может проникать дальше, в зависимости от требований к качеству выпущенной версии.*

#### 2.3.3 Регрессионное тестирование

**Регрессионное тестирование** - это вид тестирования, направленный на проверку изменений, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает, как и прежде. Регрессионными могут быть как **функциональные**, так и **нефункциональные тесты**. Как правило, для регрессионного тестирования используются тест кейсы, написанные на ранних стадиях разработки и тестирования. Это дает гарантию того, что изменения в новой версии приложения не повредили уже существующую функциональность. Рекомендуется делать автоматизацию регрессионных тестов, для ускорения последующего процесса тестирования и обнаружения дефектов на ранних стадиях разработки программного обеспечения.

Сам по себе термин "Регрессионное тестирование", в зависимости от контекста использования может иметь разный смысл. Сэм Канер, к примеру, описал 3 основных типа регрессионного тестирования:

* **Регрессия багов** (**Bug regression**) - попытка доказать, что исправленная ошибка на самом деле не исправлена
* **Регрессия старых багов** (**Old bugs regression**) - попытка доказать, что недавнее изменение кода или данных сломало исправление старых ошибок, т.е. старые баги стали снова воспроизводиться.
* **Регрессия побочного эффекта** (**Side effect regression**) - попытка доказать, что недавнее изменение кода или данных сломало другие части разрабатываемого приложения

#### 2.3.4 Санитарное тестирование

**Санитарное тестирование** или **проверка согласованности/исправности** - это узконаправленное тестирование достаточное для доказательства того, что конкретная функция работает согласно заявленным в спецификации требованиям. Является подмножеством регрессионного тестирования. Используется для определения работоспособности определенной части приложения после изменений произведенных в ней или окружающей среде. Обычно выполняется вручную. В некоторых источниках ошибочно полагают, что санитарное и дымовое тестирование - это одно и тоже. Однако, в отличие от дымового (*Smoke testing*), **санитарное тестирование** (*Sanity testing*) **направлено вглубь** проверяемой функции, в то время как **дымовое направлено вширь**, для покрытия тестами как можно большего функционала в кратчайшие сроки.

***Слайд 20***

## 3. Тестовые артефакты, тест-дизайн

*В соответствие с процессами или методологиями разработки ПО, во время проведения тестирования создается и используется определенное количество* ***тестовых артефактов*** *(документы, модели и т.д.). Наиболее распространенными тестовыми артефактами являются:*

* ***План тестирования*** – это документ описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.
* ***Набор тестовых случаев (кейсов) и тестов*** – это последовательность действий, по которой можно проверить соответствует ли тестируемая функция установленным требованиям.

*Под тестовым случаем понимается структура вида:* ***Action*** *>* ***Expected Result*** *>* ***Test Result (****passed/failed/blocked****)****. Например, «Открытие страницы авторизации» > Страница авторизации открылась > Passed. Тест кейсы разделяются по ожидаемому результату на* ***позитивные*** *(использует только корректные данные и проверяет, что приложение правильно выполнило вызываемую функцию) и* ***негативные*** *(оперирует как корректными, так и некорректными данными (минимум 1 некорректный параметр) и ставит целью проверку исключительных ситуаций (срабатывание валидаторов), а также проверяет, что вызываемая приложением функция не выполняется при срабатывании валидатора).*

***Слайд 21***

* ***Баг репорты/Дефекты*** – это документы, описывающие ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.
  + - **Структура баг репорта** – описывает основные поля баг/репорта.
    - **Серьезность и Приоритет дефектов**:

**Серьезность** (**Severity**) - это атрибут, характеризующий влияние дефекта на работоспособность приложения.

**S1 Блокирующая (Blocker)**

*Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или ее ключевыми функциями становится невозможна. Решение проблемы необходимо для дальнейшего функционирования системы.*

**S2 Критическая (Critical)**

*Критическая ошибка, неправильно работающая ключевая бизнес логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, без возможности решения проблемы, используя другие входные точки. Решение проблемы необходимо для дальнейшей работы с ключевыми функциями тестируемой системой.*

**S3 Значительная (Major)**

*Значительная ошибка, часть основной бизнес логики работает некорректно. Ошибка не критична или есть возможность для работы с тестируемой функцией, используя другие входные точки.*

**S4 Незначительная (Minor)**

*Незначительная ошибка, не нарушающая бизнес логику тестируемой части приложения, очевидная проблема пользовательского интерфейса.*

**S5 Тривиальная (Trivial)**

*Тривиальная ошибка, не касающаяся бизнес логики приложения, плохо воспроизводимая проблема, малозаметная посредствам пользовательского интерфейса, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.*

**Приоритет** (**Priority**) - это атрибут, указывающий на очередность выполнения задачи или устранения дефекта. Можно сказать, что это инструмент менеджера по планированию работ. Чем выше приоритет, тем быстрее нужно исправить дефект.

**P1 Высокий (High)**

*Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критической для проекта.*

**P2 Средний (Medium)**

*Ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения.*

**P3 Низкий (Low)**

*Ошибка должна быть исправлена, ее наличие не является критичной, и не требует срочного решения.*

Порядок исправления ошибок по их приоритетам:

**High -> Medium -> Low**

Требования к количеству открытых багов:

* Наличие открытых дефектов P1, P2 и S1, S2, считается неприемлемым для проекта. *Все подобные ситуации требуют срочного решения и идут под контроль к менеджерам проекта.*
* Наличие строго ограниченного количества открытых ошибок P3 и S3, S4, S5 не является критичным для проекта и допускается в выдаваемом приложении. *Количество же открытых ошибок зависит от размера проекта и установленных критериев качества.*
  + - ***Написание баг репорта***

***Баг репорт*** *-* ***это технический документ*** *и в связи с этим язык описания проблемы должен быть техническим. Должна использоваться правильная терминология при использовании названий элементов пользовательского интерфейса (editbox, listbox, combobox, link, text area, button, menu, popup menu, title bar, system tray и т.д.), действий пользователя (click link, press the button, select menu item и т.д.) и полученных результатах (window is opened, error message is displayed, system crashed и т.д.).*

*Очень важно четко описать все шаги, с упоминаем всех вводимых данных (имени пользователя, данных для заполнения формы) и промежуточных результатов.*

***Слайд 22***

### Тест дизайн

**Тест дизайн** – это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы), в соответствии с определёнными ранее критериями качества и целями тестирования.

План работы над тест дизайном включает следующие этапы:

* анализ имеющихся проектных артефактов: документация (спецификации, требования, планы), модели, исполняемый код и т.д.;
* написание спецификации по тест дизайну;
* проектирование и создание тестовых случаев.

За тест дизайн отвечают тест-аналитики и тест-дизайнеры. Первые определяют что тестировать, а вторые – как тестировать. *Задача тест аналитиков и дизайнеров сводится к тому, чтобы используя различные стратегии и техники дизайна создать набор тестовых случаев, обеспечивающий оптимальное тестовое покрытие тестируемого приложения. Однако, на большинстве проектов эти роли не выделяется, а доверяется обычным тестировщикам, что не всегда положительно сказывается на качестве тестов, тестировании и, как из этого следует, на качестве программного обеспечения.*

***Слайд 23***

**Тестовое покрытие**

**Тестовое Покрытие** - это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.

Если рассматривать тестирование как "проверку соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов", то именно этот конечный набор тестов и будет определять тестовое покрытие:

*Чем выше требуемый уровень тестового покрытия, тем больше тестов будет выбрано, для проверки тестируемых требований или исполняемого кода.*

*Сложность современного программного обеспечения и инфраструктуры сделало невыполнимой задачу проведения тестирования со 100% тестовым покрытием. Поэтому для разработки набора тестов, обеспечивающего более менее высокий уровень покрытия можно использовать специальные инструменты либо техники тест дизайна.*

Рассмотрим **подходы** к оценке и измерению тестового покрытия.

1. **Покрытие требований**

Расчет тестового покрытия относительно требований проводится по формуле:

где: - тестовое покрытие;

- количество требований, проверяемых тест кейсами;

- общее количество требований.

*Для измерения покрытия требований, необходимо проанализировать требования к продукту и разбить их на пункты. Опционально каждый пункт связывается с тест кейсами, проверяющими его. Совокупность этих связей - и является матрицей трассировки. Проследив связи, можно понять какие именно требования проверяет тестовый случай.*

*Тесты не связанные с требованиями не имеют смысла. Требования, не связанные с тестами - это "белые пятна", т.е. выполнив все созданные тест кейсы, нельзя дать ответ реализовано данное требование в продукте или нет.*

*Для оптимизации тестового покрытия при тестировании на основании требований, наилучшим способом будет использование стандартных техник тест дизайна.*

2. **Покрытие кода**

Расчет тестового покрытия относительно исполняемого кода программного обеспечения проводится по формуле:

где: - тестовое покрытие

- кол-ва строк кода, покрытых тестами

- общее кол-во строк кода.

*В настоящее время существует инструментарий (например:* [*Clover*](http://www.atlassian.com/software/clover/)*), позволяющий проанализировать в какие строки были вхождения во время проведения тестирования, благодаря чему можно значительно увеличить покрытие, добавив новые тесты для конкретных случаев, а также избавиться от дублирующих тестов. Проведение такого анализа кода и последующая оптимизация покрытия достаточно легко реализуется в рамках тестирования белого ящика (white-box testing) при модульном, интеграционном и системном тестировании; при тестировании же черного ящика (black-box testing) задача становится довольно дорогостоящей, так как требует много времени и ресурсов на установку, конфигурацию и анализ результатов работы, как со стороны тестировщиков, так и разработчиков.*

3. **Тестовое покрытие на базе анализа потока управления**

**Тестирование потоков управления** (Control Flow Testing) - это одна из техник тестирования белого ящика, основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для покрытия этих путей.

Фундаментом для тестирования потоков управления является построение графов потоков управления (Control Flow Graph), основными блоками которых являются:

* блок процесса - одна точка входа и одна точка выхода
* точка альтернативы - одна точка входа, две и более точки выхода
* точка соединения - две и более точек входа, одна точка выхода

***Слайд 24***

Для тестирования потоков управления определены разные **уровни тестового покрытия**:

|  |  |  |
| --- | --- | --- |
| **Уровень** | **Название** | **Краткое описание** |
| Уровень 0 | -- | Тестируется все возможные элементы ПО. |
| Уровень 1 | Покрытие операторов | Каждый оператор должен быть выполнен как минимум один раз. |
| Уровень 2 | Покрытие альтернатив / Покрытие ветвей | Каждый узел с ветвлением (альтернатива) выполнен как минимум один раз. |
| Уровень 3 | Покрытие условий | Каждое условие, имеющее TRUE и FALSE на выходе, выполнено как минимум один раз. |
| Уровень 4 | Покрытие условий альтернатив | Тестовые случаи создаются для каждого условия и альтернативы |
| Уровень 5 | Покрытие множественных условий | Достигается покрытие альтернатив, условий и условий альтернатив (Уровни 2, 3 и 4) |
| Уровень 6 | “Покрытие бесконечного числа путей” | Если, в случае зацикливания, количество путей становится бесконечным, допускается существенное их сокращение, ограничивая количество циклов выполнения, для уменьшения числа тестовых случаев. |
| Уровень 7 | Покрытие путей | Все пути должны быть проверены |

Основываясь на данных этой таблицы, вы сможете спланировать необходимый уровень тестового покрытия, а также оценить уже имеющийся.

***Слайд 25***

**Техники тест дизайна**

*Многие люди тестируют и пишут* [*тестовые случаи (test cases)*](http://www.protesting.ru/testing/testcase.html)*, но не многие пользуются специальными* ***техниками тест дизайна****. Постепенно, набираясь опыта они осознают, что постоянно делают одну и ту же работу, поддающуюся конкретным правилам. И тогда они находят, что все эти правила уже описаны.*

Краткое описание наиболее распространенных техник тест дизайна:

* **Эквивалентное Разделение** (**Equivalence Partitioning - EP**). *Как пример, у вас есть диапазон допустимых значений от 1 до 10, вы должны выбрать одно верное значение внутри интервала, скажем, 5, и одно неверное значение вне интервала - 0.*
* **Анализ Граничных Значений** (**Boundary Value Analysis - BVA**). *Если взять пример выше, в качестве значений для позитивного тестирования выберем минимальную и максимальную границы (1 и 10), и значения больше и меньше границ (0 и 11). Анализ Граничный значений может быть применен к полям, записям, файлам, или к любого рода сущностям имеющим ограничения.*
* **Причина / Следствие** (**Cause/Effect - CE**). *Это, как правило, ввод комбинаций условий (причин), для получения ответа от системы (Следствие).* *Например, вы проверяете возможность добавлять клиента, используя определенную экранную форму. Для этого вам необходимо будет ввести несколько полей, таких как "Имя", "Адрес", "Номер Телефона" а затем, нажать кнопку "Добавить" - эта "Причина". После нажатия кнопки "Добавить", система добавляет клиента в базу данных и показывает его номер на экране - это "Следствие".*
* **Предугадывание ошибки** (**Error Guessing - EG**). *Это когда тест аналитик использует свои знания системы и способность к интерпретации спецификации на предмет того, чтобы "предугадать" при каких входных условиях система может выдать ошибку.* ***Например,*** *спецификация говорит: "пользователь должен ввести код". Тест аналитик, будет думать: "Что, если я не введу код?", "Что, если я введу неправильный код? ", и так далее. Это и есть предугадывание ошибки.*
* **Исчерпывающее тестирование** (**Exhaustive Testing - ET**) - *это крайний случай. В пределах этой техники вы должны проверить все возможные комбинации входных значений, и в принципе, это должно найти все проблемы. На практике применение этого метода не представляется возможным, из-за огромного количества входных значений.*

***Слайд 26***

## Литература:

1. Орлов С.А. Технологии разработки программного обеспечения : учеб. для вузов. - 3-е изд СПб.: Питер, 2004
2. Аллен Э. Типичные ошибки проектирования : [Пер. с англ.] СПб.: Питер, 2003
3. Фатрелл Р.Т. Управление программными проектами: достижение оптимального качества при минимуме затрат : [Пер. с англ.] М.: Вильямс, 2003
4. Липаев В.В. Надежность программных средств./ В.В. Липаев — М.: СИНТЕГ,1998.

Майерс Г. Надежность программного обеспечения: Пер. с англ/ Г. Майерс.. - М.: Мир, 1980.