Билеты к зачету по тестированию

1. **Определение «Качество». Перечислить основные характеристики качества. Характеристика «Функциональность (Functionality)».**
2. **Определение Качества. Перечислить основные характеристики качества. Характеристика «Надежность (Reliability)».**
3. **Определение «Качество». Перечислить основные характеристики качества. Характеристика «Удобство использования (Usability)».**
4. **Определение Качества. Перечислить основные характеристики качества. Характеристика «Эффективность (Efficiency)».**
5. **Определение «Качество». Перечислить основные характеристики качества. Характеристика «Удобство сопровождения (Maintainability)».**
6. **Определение Качества. Перечислить основные характеристики качества. Характеристика «Портативность (Portability)».**
7. **Определение «Качество». Перечислить основные характеристики качества. Качественное и количественное выражение.**

Качество системы - это степень удовлетворения системой заявленных и подразумеваемых потребностей различных заинтересованных сторон, которая позволяет, таким образом, оценить достоинства.

Функциональность (Functionality) - определяется способностью ПО решать задачи, которые соответствуют зафиксированным и предполагаемым потребностям пользователя, при заданных условиях использования ПО. Т.е. эта характеристика отвечает за то, что ПО работает исправно и точно, функционально совместимо, соответствует стандартам отрасли и защищено от несанкционированного доступа.

Надежность (Reliability) – способность ПО выполнять требуемые задачи в обозначенных условиях на протяжении заданного промежутка времени или указанное количество операций. Атрибуты данной характеристики – это завершенность и целостность всей системы, способность самостоятельно и корректно восстанавливаться после сбоев в работе, отказоустойчивость.

Удобство использования (Usability) – возможность легкого понимания, изучения, использования и привлекательность ПО для пользователя.

Эффективность (Efficiency) – способность ПО обеспечивать требуемый уровень производительности в соответствии с выделенными ресурсами, временем и другими обозначенными условиями.

Удобство сопровождения (Maintainability) – легкость, с которой ПО может анализироваться, тестироваться, изменяться для исправления дефектов, для реализации новых требований, для облегчения дальнейшего обслуживания и адаптироваться к имеющемуся окружению.

Портативность (Portability) – характеризует ПО с точки зрения легкости его переноса из одного окружения (software/hardware) в другое.

Важно уметь качественные характеристики сводить к количественным. Это позволяет контролировать прогресс с развитием продукта.

8. Определение «Тестирование». Определение «Дефект». Отличие терминов «Контроль качества (Quality Control)» и «Обеспечение качества (Quality Assurance)».

Тестирование – это проведение испытаний продукта для выявления наличия требуемой функциональности, соответствия требуемым характеристикам и обнаружения особенностей продукта.

Особенность продукта - несоответствие ожидаемому поведению, возникшее из-за ограничений алгоритмов (особенность реализации), несовершенства инструментов разработки и влияния параметров среды на продукт.

Дефект – несоответствие требуемым характеристикам, то есть поведение программы, которое явно противоречит требованию, предъявляемому к продукту.

Quality Assurance (обеспечение качества) - это превентивный процесс, задачей которого является обеспечение качества продукта в будущем. В этом смысле Quality Assurance более ориентирован на процесс.

Quality Control (контроль качества) - это процесс нахождения ошибок в продукте, с целью их последующего исправления. Задачей Quality Control является поддержка качества продукта в текущий момент времени. Quality Control ориентирован на продукт, разрабатываемый в данный момент.

Кратко: первый направлен на контроль (проверку, измерение) характеристик, а второй – на улучшение процессов, чтобы получать требуемое качество.

Quality Assurance включает в себя Quality Control наряду с другими процессами по улучшению качества работы компании.

Говоря другими словами, Quality Assurance гарантирует, что процесс поставлен правильно и дает предсказуемый результат, в то время как Quality Control гарантирует, что продукт удовлетворяет указанному набору требований.

9. Экономическое обоснование процесса тестирования. Следствия.

$$П = \sum p_i * c_i$$

П – общие потери

c_i – стоимость сценария для пользователя

p_i – вероятность ошибки на сценарии

Стоимость сценария для случая с программным обеспечением обычно составляет стоимость контракта.

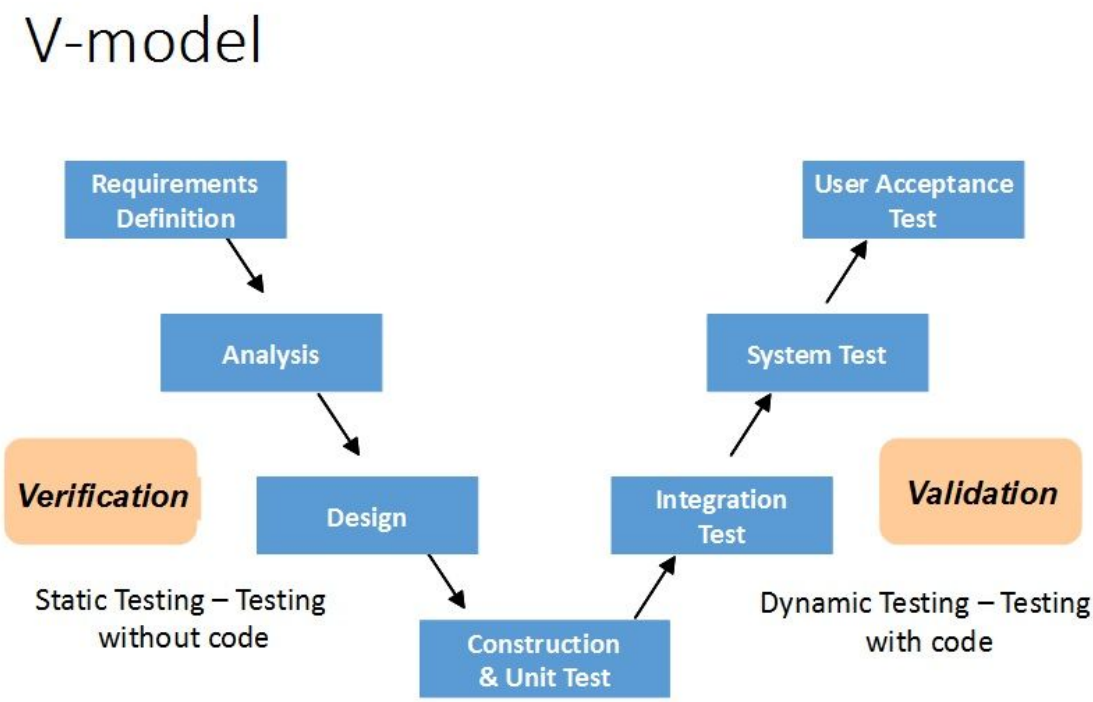
Индекс i является номером сочетания системного сценария, входных данных и среды, где функционирует система. (Т.к. речь идет о пользователе, то можно говорить о бизнес сценарии, но в контексте процессов тестирования удобнее говорить о системных.)

Из формулы следует, что вероятность ошибки можно свести к нулю только если будем знать четкий пользовательский сценарий, среду (система и другой софт) и иметь пользовательские данные.

С другой стороны, если от продукта нет выгоды, то тестировать его не имеет смысла. Тут стоит отделять понятия бесплатный продукт, отсутствие прибыли и отсутствие выгоды. Во втором случае могут быть имиджевые риски, а в первом – вполне может быть прибыль (реклама).

10. V-модель. Определение «Верификация (Verification)».

11. V-модель. Определение «Валидация (Verification)».



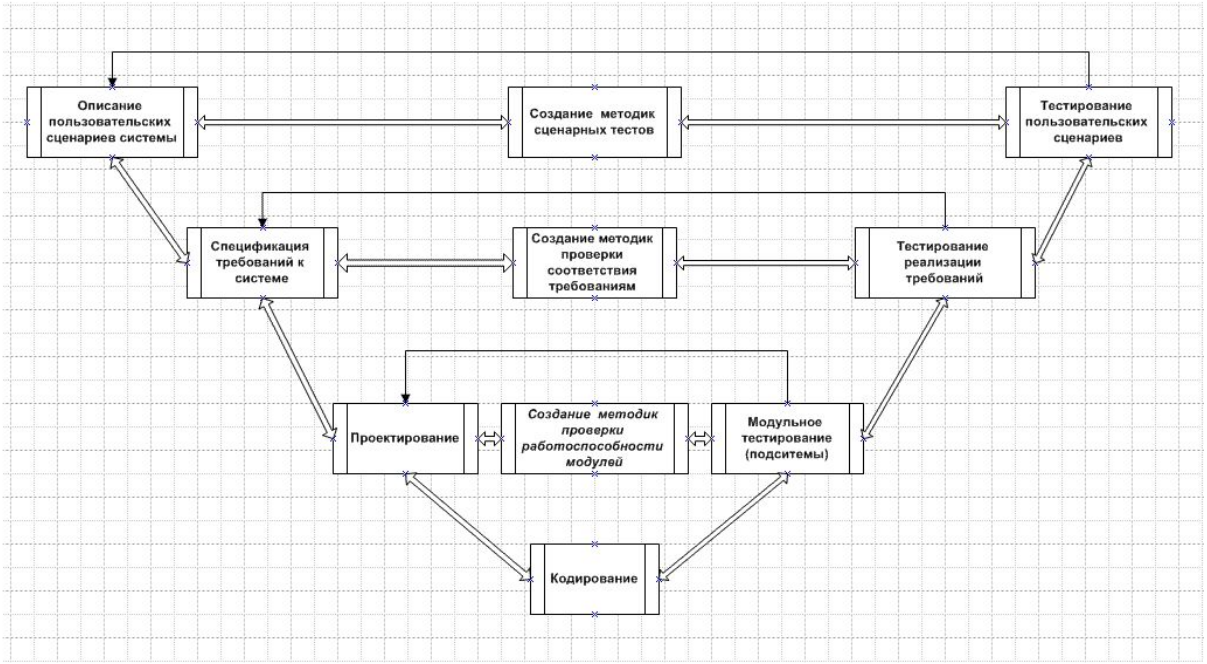
Верификация (verification) – это процесс оценки системы или её компонентов с целью определения того, удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа. То есть, выполняются ли задачи, цели и сроки по разработке продукта.

Валидация (validation) – это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

12. Основная документация в тестировании.

В соответствии 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** (Test Plan) - 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 тест кейсов и тестов** (Test Case & Test suite) - это последовательность действий, по которой можно проверить соответствует ли тестируемая функция установленным требованиям.
- **Дефекты / Баг Репорты** (Bug Reports / Defects) - это документы, описывающие ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.



Кратко:

- **Стратегия тестирования**: Какие ограничения?
- **План тестирования**: Что делать?
- **Методика тестирования**: Как делать?
- **Отчет о тестировании**: Что наделали?

13. Основные требования к средству автоматизации тестирования. Фундаментальная проблема автоматизации.

Требования к средствам автоматизированного тестирования:

- Стабильность
- Функциональность
- Удобство. Создание нового. Модификация. Рефакторинг.

Фундаментальная проблема автоматизации: валидация результатов.

14. Виды тестирования. Функциональное тестирование.

15. Виды тестирования. Тестирование производительности.

16. Виды тестирования. Юзабилити-тестирование.

17. Виды тестирования. Тестирование безопасности.

18. Виды тестирования. Тестирование локализации.

- **Функциональное тестирование**
- **Нефункциональное тестирование**
    - Тестирование производительности
        - Тестирование стабильности
        - Нагрузочное тестирование
        - Стресс-тестирование
    - Юзабилити-тестирование
    - Тестирование безопасности

- Тестирование локализации

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

Преимущества функционального тестирования:

- имитирует фактическое использование системы;

Недостатки функционального тестирования:

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

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

Основные виды тестирования производительности:

- **Тестирование производительности (Performance testing)**

Задачей тестирования производительности 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
- **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.
- **Объемное тестирование (Volume Testing)**

Задачей объемного тестирования является получение оценки производительности при увеличении объемов данных в базе данных приложения, при этом происходит:

  - измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
  - может производиться определение количества пользователей, одновременно работающих с приложением
- **Тестирование стабильности или надежности (Stability / Reliability Testing)**

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.

16. **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*)

17. **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.

Принципы безопасности программного обеспечения:

1. конфиденциальность
2. целостность
3. доступность

Конфиденциальность - это сокрытие определенных ресурсов или информации.

Целостность:

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, насколько важной является процедура восстановления данных.

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.

18. Тестирование локализации - это процесс тестирования локализованной версии программного продукта. Проверка правильности перевода элементов интерфейса пользователя, проверка правильности перевода системных сообщений и ошибок, проверка перевода раздела "Помощь"/"Справка" и сопроводительной документации.

С помощью тестирования локализации проверяются перевод, адаптация элементов интерфейса, вспомогательные файлы, правильное обоснование и адаптация элементов интерфейса, а также правила написания текста.

Цель теста локализации – убедиться, что приложение поддерживает многоязыковый интерфейс и функции. А также проблемы связанные с локализацией (перевод на другой язык, формат дат и чисел, почтовые адреса, порядок имени и фамилии, валюты и т.д.). Орфография и грамматика обычно не тестируются.

Локализация программного обеспечения - процесс адаптации к культурным особенностям той или иной страны: перевод документации, элементов пользовательского интерфейса, вспомогательных материалов с одного языка на другой.

Процесс локализации тестирования может включать в себя:

- Определение и изучение списка поддерживаемых языков
- Проверка правильности перевода согласно тематике данного сайта или программы
- Проверка правильности перевода элементов интерфейса пользователя
- Проверка правильности перевода системных сообщений и ошибок
- Проверка перевода раздела «Помощь» и сопроводительной документации

19. Принципы тестирования. Белый ящик. Критерий окончания.
20. Принципы тестирования. Черный ящик. Критерий окончания.

Принципы тестирования:

- Тестирование чёрного ящика
- Тестирование белого ящика
- Тестирование серого ящика

Black Box

Summary: Мы не знаем, как устроена тестируемая система.
Тестирование методом «черного ящика», также известное как тестирование, основанное на спецификации или тестирование поведения – техника тестирования, основанная на работе исключительно с внешними интерфейсами тестируемой системы.
Тестирование черного ящика – это:
– тестирование, как функциональное, так и нефункциональное, не предполагающее знания внутреннего устройства компонента или системы.
– тест-дизайн, основанный на технике черного ящика – процедура написания или выбора тест-кейсов на основе анализа функциональной или нефункциональной спецификации компонента или системы без знания ее внутреннего устройства.
Почему именно «черный ящик»? Тестируемая программа для тестировщика – как черный непрозрачный ящик, содержания которого он не видит. Целью этой техники является поиск ошибок в таких категориях:
– неправильно реализованные или недостающие функции;
– ошибки интерфейса;

– ошибки в структурах данных или организации доступа к внешним базам данных;
– ошибки поведения или недостаточная производительности системы;
Таким образом, мы не имеем представления о структуре и внутреннем устройстве системы. Нужно концентрироваться на том, *что* программа делает, а не на том, *как* она это делает.

Поскольку это *тип* тестирования, по определению он может включать другие его виды. Тестирование черного ящика может быть как функциональным, так и нефункциональным. Функциональное тестирование предполагает проверку работы функций системы, а нефункциональное – соответственно, общие характеристики нашей программы.

Преимущества:
– тестирование производится с позиции конечного пользователя и может помочь обнаружить неточности и противоречия в спецификации;
– тестировщику нет необходимости знать языки программирования и углубляться в особенности реализации программы;
– тестирование может производиться специалистами, независимыми от отдела разработки, что помогает избежать предвзятого отношения;
– можно начинать писать тест-кейсы, как только готова спецификация.

Недостатки:
– тестируется только очень ограниченное количество путей выполнения программы;
– без четкой спецификации (а это скорее реальность на многих проектах) достаточно трудно составить эффективные тест-кейсы;
– некоторые тесты могут оказаться избыточными, если они уже были проведены разработчиком на уровне модульного тестирования;
Противоположностью техники черного ящика является тестирование методом белого ящика, речь о котором пойдет ниже.

White Box

Summary: Нам известны все детали реализации тестируемой программы.
Тестирование методом белого ящика (также: прозрачного, открытого, стеклянного ящика; основанное на коде или структурное тестирование) – метод тестирования программного обеспечения, который предполагает, что внутренняя структура/устройство/реализация системы известны тестировщику. Мы выбираем входные значения, основываясь на знании кода, который будет их обрабатывать. Точно так же мы знаем, каким должен быть результат этой обработки. Знание всех особенностей тестируемой программы и ее реализации – обязательны для этой техники. Тестирование белого ящика – углубление во внутреннее устройство системы, за пределы ее внешних интерфейсов.

Тестирование белого ящика – это:
– тестирование, основанное на анализе внутренней структуры компонента или системы.
– тест-дизайн, основанный на технике белого ящика – процедура написания или выбора тест-кейсов на основе анализа внутреннего устройства системы или компонента.

Техника белого ящика применима на разных уровнях тестирования – от модульного до системного, но главным образом применяется именно для реализации модульного тестирования компонента его автором.

Преимущества:
– тестирование может производиться на ранних этапах: нет необходимости ждать создания пользовательского интерфейса;
– можно провести более тщательное тестирование, с покрытием большого количества путей выполнения программы.

Недостатки:
– для выполнения тестирования белого ящика необходимо большое количество специальных знаний

– при использовании автоматизации тестирования на этом уровне, поддержка тестовых скриптов может оказаться достаточно накладной, если программа часто изменяется.

Сравнение Black Box и White Box:

Критерий	Black Box	White Box
Определение	тестирование, как функциональное, так и нефункциональное, не предполагающее знания внутреннего устройства компонента или системы	тестирование, основанное на анализе внутренней структуры компонента или системы
Уровни, к которым применима техника	В основном: • Приемочное тестирование • Системное тестирование	В основном: • Юнит-тестирование • Интеграционное тестирование
Кто выполняет	Как правило, тестировщики	Как правило, разработчики
Знание программирования	Не нужно	Необходимо
Знание реализации	Не нужно	Необходимо
Основа для тест-кейсов	Спецификация, требования	Проектная документация

21. Объекты тестирования.

Объекты тестирования:

- **Тестирование компонентов (модулей)**

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. Все найденные дефекты, как правило исправляются в коде без формального их описания в системе менеджмента багов (Bug Tracking System).

- **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** (*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 (см. также Integration testing - Bottom Up)

- **Сверху вниз** (*Top Down Integration*)

В начале 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. (см. также Top Down Integration)

- **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 неверно, то сам процесс интеграции сильно осложнится, что станет преградой для команды тестирования при достижении основной цели интеграционного тестирования (см. также Integration testing - Big Bang)

- **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, не предусмотренные 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 (test cases), 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 тест кейсы (test cases), которые должны быть протестированы.

22. Дефект. Стандартный жизненный цикл.

Дефект/Ошибка (Defect/Bug/Reclamation) – это документ, описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

Структура:

- Короткое описание
- Детальное описание
- Последовательность шагов для воспроизведения

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

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.

Градация Серьезности дефекта (Severity)

- **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.

Градация Приоритета дефекта (Priority)

- **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.