Концептуальная основа гарантированного управления качеством

﻿﻿-Основа управления качеством - планирование разноаспектной деятельности, связанной с реализацией проекта, стандартизация всех фаз жизненного цикла

﻿﻿-Планы разрабатываются в соответствие с требованиями стандартов

﻿﻿-Процедуры выполняются в соответствие с планами

﻿﻿-Продукты используются в соответствие со стандартами

**22. Способы тестирования программных систем. Понятие альфа- и бета тестирования**

Тестирование программных систем — это процесс проверки и оценки программного обеспечения (ПО) для выявления ошибок, недочетов и соответствия требованиям пользователя. Программное тестирование можно разделить на различные подходы и методы. Основные способы включают:

- Модульное тестирование (Unit Testing): проверка отдельных модулей или компонентов программы.

- Интеграционное тестирование (Integration Testing): проверка взаимодействия между различными модулями.

- Системное тестирование (System Testing): проверка всей системы в целом, чтобы убедиться, что она отвечает заданным требованиям.

- Регрессионное тестирование (Regression Testing): проверка исправленной программы на предмет появления новых ошибок.

- Приёмочное тестирование (Acceptance Testing): проверка системы пользователем перед запуском в производство.

- Альфа- и бета-тестирование: фокус на пользовательском тестировании продукта на этапе завершения разработки.

**Альфа-тестирование**

Альфа-тестирование — это этап тестирования программного обеспечения, который проводится внутри компании-разработчика перед его передачей внешним пользователям. Его основная цель — выявление возможных дефектов до выпуска продукта.

Основные характеристики:

- Выполняется тестировщиками или разработчиками внутри компании.

- Может включать моделирование реальных условий использования.

- Фокус на функциональности, производительности и выявлении критических ошибок.

- Простейшее описание: это проверка "в лабораторных условиях".

Этапы:

1. Проверка отдельных функций и компонентов.

2. Интеграционное тестирование (взаимодействие всех частей системы).

3. Полное системное тестирование.

Преимущества:

- Позволяет обнаружить максимальное количество багов внутри компании.

- Ранние этапы позволяют своевременно исправлять ошибки.

Ограничения:

- Не всегда отражает поведение реальных пользователей.

- Недостаточная проработка реальных условий эксплуатации.

**Бета-тестирование**

Бета-тестирование — это этап тестирования программного обеспечения, который выполняется потенциальными или реальными пользователями за пределами компании-разработчика. Этот этап начинается после успешного альфа-тестирования.

Основные характеристики:

- Проводится в реальных условиях эксплуатации продукта.

- Привлекаются конечные пользователи, которые предоставляют обратную связь.

- Часто на данном этапе выявляются ошибки, связанные с реальными сценариями использования.

Этапы:

1. Распространение ограниченной версии программного обеспечения среди выбранной группы пользователей.

2. Сбор обратной связи (баги, предложения по улучшению, отзывы о производительности).

3. Внесение доработок и подготовка к финальному релизу.

Преимущества:

- Тестируется в среде реальной эксплуатации.

- Учитываются запросы и пожелания клиентов.

- Стимулирует развитие пользовательской базы и доверие пользователей.

Ограничения:

- Возможны недовольства пользователей, если продукт нестабилен.

- Могут быть пропущены сложные баги, так как пользователи сфокусированы на стандартных задачах.

**23. Концептуальная основа тестирования (80-е годы). Сценарное тестирование.**

*Презентация:*

Краткий очерк истории тестирования  
80-е годы  
1. Ключевое изменение места тестирования в разработке ПО: вместо финальной стадии реализации проекта тестирование стало применяться в течение всего жизненного цикла разработки, что позволило не только сократить «латентный период» ошибки, но и в известной степени предотвратить их появление  
2. Бурное развитие и формализация методов тестирования  
2. Первые попытки автоматизации тестирования

*Учебник из инета:*

В 80-х годах произошло ключевое изменение места тестирования в разработке ПО: вместо одной из финальных стадий создания проекта тестирование стало применяться на протяжении всего цикла разработки (software lifecycle7), что позволило в очень многих случаях не только быстро обнаруживать и устранять проблемы, но даже предсказывать и предотвращать их появление. В этот же период времени отмечено бурное развитие и формализация методологий тестирования и появление первых элементарных попыток автоматизировать тестирование.

Лекция

Сценарное тестирование-классическое тестирование по предварительно написанным и задокументированным сценариям.

В пользу сценарного тестирования: сравнительная легкость планирования: тест-кейсы можно легко поделить между различными тестировщиками или командами.

Что такое сценарное тестирование?

Сценарное тестирование — это метод тестирования программного обеспечения, который включает в себя создание сценариев или последовательности действий для проверки функциональности приложения. Эти сценарии представляют собой реальные ситуации использования, что помогает тестировщикам понять, как приложение будет работать в руках конечного пользователя.

Зачем нужно сценарное тестирование?

Сценарное тестирование помогает:

- Проверить взаимодействие различных функций приложения;

- Обнаружить потенциальные ошибки, неявно возникающие при использовании приложения в реальных условиях;

- Улучшить пользовательский опыт за счёт выявления и исправления неудобных процессов.

Как проводится сценарное тестирование?

1. Определение сценариев: тестировщики разрабатывают сценарии на основе предполагаемых действий пользователя.

2. Подготовка тестовых данных: сборка данных, необходимых для выполнения сценария.

3. Выполнение теста: следование каждому шагу сценария и запись результатов.

4. Анализ: оценка результатов тестирования, выявление ошибок и их документирование.

Таким образом, сценарное тестирование ориентировано на пользователя и направлено на улучшение качества программного обеспечения путём выявления проблемных моментов в реальных пользовательских сценариях.

**24. Концептуальная основа тестирования (80-е годы). Регрессионное тестирование**

*Презентация:*

Краткий очерк истории тестирования  
80-е годы  
1. Ключевое изменение места тестирования в разработке ПО: вместо финальной стадии реализации проекта тестирование стало применяться в течение всего жизненного цикла разработки, что позволило не только сократить «латентный период» ошибки, но и в известной степени предотвратить их появление  
2. Бурное развитие и формализация методов тестирования  
2. Первые попытки автоматизации тестирования

*Учебник из инета:*

В 80-х годах произошло ключевое изменение места тестирования в разработке ПО: вместо одной из финальных стадий создания проекта тестирование стало применяться на протяжении всего цикла разработки (software lifecycle7), что позволило в очень многих случаях не только быстро обнаруживать и устранять проблемы, но даже предсказывать и предотвращать их появление. В этот же период времени отмечено бурное развитие и формализация методологий тестирования и появление первых элементарных попыток автоматизировать тестирование.

Регрессионное тестирование — это тип тестирования программного обеспечения, направленный на проверку того, что внесённые изменения (например, исправления багов, добавление новой функциональности или обновления) не вызвали появление новых ошибок или не нарушили уже работающие функции.

Это критически важный этап тестирования, особенно в случае частых обновлений приложений, чтобы сохранить их стабильность и качество.

Основные цели регрессионного тестирования

1. Проверить, что новые изменения не оказали негативного влияния на существующий функционал.

2. Убедиться, что исправленные ошибки не возникли снова (проверка на рецидив).

3. Поддерживать общее качество приложения после обновлений.

Когда выполняется регрессионное тестирование?

- После исправления багов;

- После изменения кода (добавления новых функций или рефакторинга);

- После обновления внешних библиотек, зависимостей или платформ.

Методы регрессионного тестирования

1. Повторное тестирование: выполнение всех ранее выполненных тестов для проверки стабильности.

2. Выборочное тестирование: проверка только тех тестов, которые затрагивают обновлённый или зависимый функционал.

3. Автоматизация: использование автоматизированных тестов для регулярной проверки стабильности приложения. Это помогает экономить время при частых обновлениях ПО.

Преимущества регрессионного тестирования

- Повышает уверенность в стабильности системы;

- Помогает избежать деградации качества продукта;

- Уменьшает риск появления неожиданных багов после обновлений.

Таким образом, регрессионное тестирование является ключевым инструментом для постоянного контроля качества ПО в условиях динамичного изменения кода.

**25. Концептуальная основа тестирования (90-е годы). Петля обратной связи как инструмент контроля реализации проекта**

****

На изображении представлена диаграмма, иллюстрирующая процесс контроля проекта с использованием петли обратной связи.

▎Основные элементы диаграммы:

1. Входные данные:

- Требования пользователя и Стандарты — это исходные данные для разработки планов проекта.

2. Блок разработки планов:

- Анализ требований пользователя и стандартов для создания четырех планов:

- SPMP (Software Project Management Plan) — план управления проектом.

- SCMP (Software Configuration Management Plan) — план управления конфигурацией.

- SVVP (Software Verification and Validation Plan) — план верификации и валидации.

- SQAP (Software Quality Assurance Plan) — план обеспечения качества.

3. Наблюдение:

- Процесс мониторинга всех этапов проекта, включающий контроль за выполнением созданных планов.

4. Создание продукта:

- Разработка самого продукта на основе созданных планов программного обеспечения (SPMP, SCMP, SVVP, SQAP).

- Периодическая отчетность о прогрессе и статусе выполнения плана.

5. Результаты:

- Продукты — конечные продукты проекта.

- Отчеты — документы, отражающие состояние проекта и результаты тестирования.

Цель петли обратной связи:

- Обеспечение постоянного контроля и корректировки разработки продукта в реальном времени, чтобы соответствовать требованиям пользователей и установленным стандартам.

- Надзор за качеством и соответствием требований каждой стадии проектирования и разработки.

Эта диаграмма демонстрирует важность структурированного подхода и мониторинга на всех этапах жизненного цикла проекта для обеспечения успешной реализации и высокого качества конечного продукта.

Петля обратной связи — это метод управления и анализа проекта, который основывается на регулярной проверке результатов, получении отзывов от заинтересованных сторон и последующем внесении корректировок. Она является важным инструментом контроля при реализации проектов, включая разработку программного обеспечения.

Такая методология применяется для адаптации работы команды, улучшения процессов и своевременного обнаружения ошибок или отклонений от целей проекта.

Основные компоненты петли обратной связи:

1. Сбор данных

На этом этапе фиксируется фактическое состояние проекта. Это могут быть ключевые метрики, прогресс задач, результаты тестов и обратная связь от команды или пользователей.

2. Анализ

Проводится оценка собранной информации. Здесь важно выяснить, соответствует ли текущий процесс или результат установленным требованиям.

3. Реакция (корректировка)

На основании анализа принимаются решения: корректировка задач, изменение подходов к разработке, внесение улучшений или усиление контроля над определёнными процессами.

4. Повтор цикла

Петля обратной связи постоянно повторяется в течение всего жизненного цикла проекта для адаптации к возникающим изменениям.

Преимущества петли обратной связи

- Обеспечивает прозрачность процессов разработки.

- Позволяет выявлять отклонения от цели на ранних этапах.

- Ускоряет принятие решений благодаря своевременной обратной связи.

- Способствует обучению команды и повышению профессионализма.

- Уменьшает риски несоответствия продукта ожиданиям пользователей или заказчиков.

Петля обратной связи, будучи итеративным процессом, помогает строить более эффективную и гибкую разработку. Она даёт команде возможность непрерывно улучшаться и адаптироваться к новым вызовам и требованиям.

**26. Ad-hoc тестирование**

Лекция

Свободное тестирование (ad-hoc testing) - это вид тестирования, который выполняется без подготовки к тестированию продукта, без определения ожидаемых результатов, проектирования тестовых сценариев. Это неформальное, импровизационное тестирование. Такой способ тестирования в большинстве случаев дает большее количество заведенных отчётов об ошибке. Это обусловлено тем, что тестировщик на первых шагах приступает к тестированию основной функциональной части продукта и выполняет как позитивные, так и негативные варианты возможных сценариев.

Основные характеристики Ad-hoc тестирования:

1. Отсутствие плана:

Тестировщик не придерживается фиксированного набора шагов или сценариев, а ищет проблемы на свое усмотрение, ориентируясь на свое знание функциональности системы.

2. Гибкость:

Ad-hoc тестирование позволяет сосредоточиться на областях, которые могут быть потенциально проблемными, не отвлекаясь на формальности.

3. Творческий подход:

Тестировщик использует свою интуицию и опыт, чтобы находить нестандартные или скрытые баги.

4. Цель — обнаружение критических ошибок:

Оно часто используется для выявления дефектов, которые могут быть пропущены при формальном тестировании.

▎Когда применяется Ad-hoc тестирование:

- Ограниченные сроки: Когда нужно быстро проверить функциональность, но времени на написание сценариев нет.

- Последняя проверка: Перед релизом, чтобы в свободной форме убедиться, что приложение стабильно.

- Непредвиденные случаи: Чтобы протестировать новую или неожиданную проблему, о которой стало известно в последний момент.

▎Преимущества Ad-hoc тестирования:

- Позволяет находить дефекты, которые невозможно выявить с помощью автоматизированного или регламентированного тестирования.

- Экономия времени, так как нет необходимости в написании тест-кейсов или документации.

- Может быть выполнено любым членом команды.

▎Недостатки Ad-hoc тестирования:

- Невоспроизводимость: Если найденный баг сложно воспроизвести, из-за отсутствия четкой документации его устранение может затрудниться.

- Зависимость от тестировщика: Качество зависит от опыта и знаний человека, проводящего тестирование.

- Ограниченный охват: Из-за отсутствия структуры часть функциональности может быть не проверена.

Ad-hoc тестирование важно как дополнительный этап, но оно не заменяет традиционное структурированное тестирование и автоматизацию.