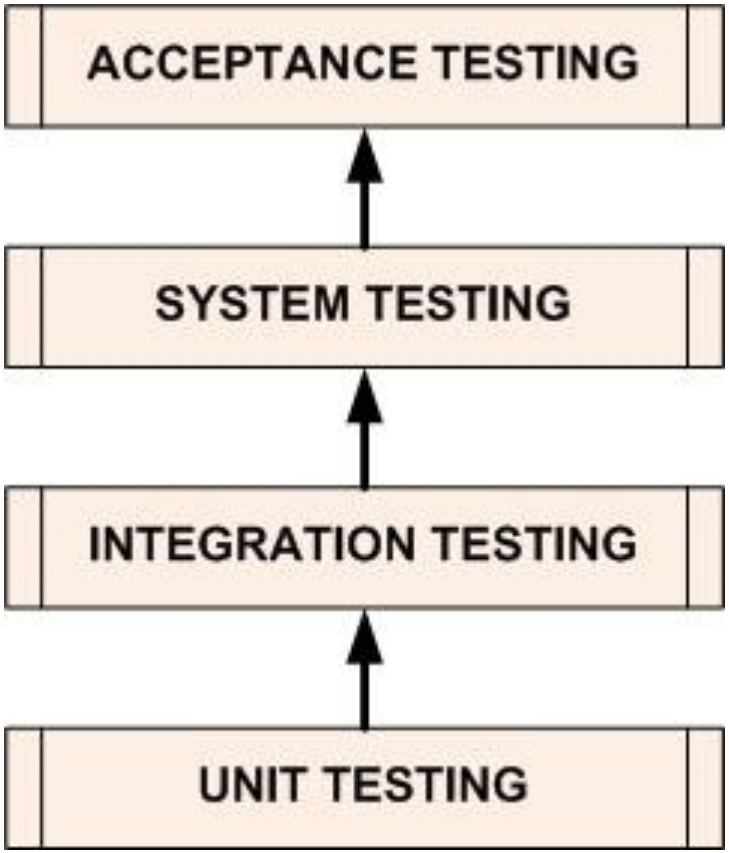
Интеграционное и системное тестирование

Уровни тестирования

В тестировании ПО можно выделить 4 уровня тестирования:



Модульное тестирование *(Unit Testing)*

Модульное тестирование заключается в тестировании отдельного модуля, как части программы, подразумевая что это только модуль и он не может существовать самостоятельно и является частью приложения, программы.

Интеграционное тестирование *(Integration Testing)*

Следующий уровень тестирования, который проводится после модульного тестирования.

После того как отдельные модули приложения были протестированы, следует провести **интеграционное тестирование**, чтобы убедиться что модули успешно функционируют в связке друг с другом.

Иными словами тестируем два и более связанных модуля, с тем чтобы проверить что интеграция прошла успешно и без явных багов.

Системное Тестирование *(System Testing)*

Уровень тестирования в котором проводят тестирование целой системы или приложения, которое было полностью разработано и которое уже готово к потенциальному релизу.

На этом уровне тестируют систему, приложение в целом, проводят тестирование на всех требуемых браузерах (*Web*-приложение) или операционных системах (*Desktop*-приложение) и проводят все требуемые типы тестирования такие как: функциональное, тестирование безопасности, тестирование *usability* (удобство использования), тестирование производительности, нагрузочное тестирование и т.д.

Приемочное Тестирование *(Acceptance Testing)*

После успешного завершения системного тестирования, продукт проходит уровень **приемочного тестирования**, который обычно проводится заказчиком или любыми другими заинтересованными лицами, с целью убеждения что продукт выглядит и работает так как требовалось изначально и было описано в требованиях к продукту.

Приемочное тестирование также может проводиться после каждого из вышеописанных уровней тестирования.

Интеграционное тестирование

Интеграционное тестирование. Задача. Цель.

Интеграционное тестирование (тестирование сборки) - тестирование части системы, состоящей из двух и более модулей.

Основная **задача** - поиск дефектов, связанных с ошибками в реализации и интерпретации взаимодействия между модулями.

Цель - удостовериться в корректности совместной работы компонент системы.

Интеграционное тестирование.

Интеграционное тестирование называют еще тестированием архитектуры системы.

С одной стороны, это название обусловлено тем, что интеграционные тесты включают в себя проверки всех возможных видов взаимодействий между программными модулями и элементами, которые определяются в архитектуре системы - таким образом, интеграционные тесты проверяют полноту взаимодействий в тестируемой реализации системы.

С другой стороны, результаты выполнения интеграционных тестов - один из основных источников информации для процесса улучшения и уточнения архитектуры системы, межмодульных и межкомпонентных интерфейсов. Т.е., с этой точки зрения, интеграционные тесты проверяют корректность взаимодействия компонент системы.

Интеграционное тестирование. Методы сборки модулей

Интеграционное тестирование использует модель «белого ящика» на модульном уровне.

Т.к. разработчику (который проводит тестирование) текст программы известен с детальностью до вызова всех модулей, входящих в тестируемый комплекс, применение структурных критериев на данном этапе возможно и оправданно.

Интеграционное тестирование применяется на этапе сборки модульно оттестированных модулей в единый комплекс.

Существуют два метода сборки модулей:

- Монолитный
- Инкрементальный

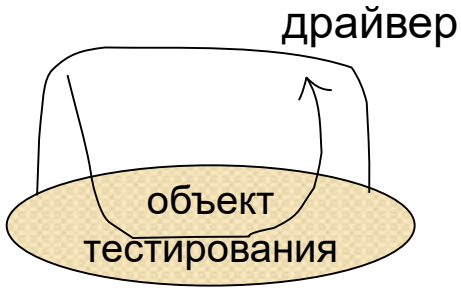
Монолитный метод (монолитное тестирование)

- Характеризуется одновременным объединением всех модулей в единый комплекс.
- Предполагает, что отдельные компоненты системы серьезного тестирования не проходили.

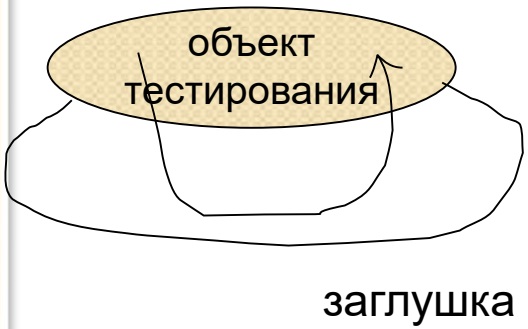
Основное преимущество данного метода - отсутствие необходимости в разработке тестового окружения, драйверов и заглушек. После разработки всех модулей выполняется их интеграция, затем система проверяется вся в целом.

Заглушки и тест-драйверы

- **Тест-драйвер** подменяет ту часть системы (или внешней среды), которая вызывает объект тестирования
  - обеспечивает передачу входных данных в объект тестирования
  - отображает результаты вызова или проверяет их правильность



- **Заглушка** подменяет часть системы, вызываемую объектом тестирования
  - имеет тот же интерфейс (API), что и подменяемая часть
  - возвращает результат того же типа



Монолитный метод (монолитное тестирование). Недостатки

- **Очень трудно выявить источник ошибки** (идентифицировать ошибочный фрагмент кода). В большинстве модулей следует предполагать наличие ошибки. Проблема сводится к определению того, какая из ошибок во всех вовлечённых модулях привела к полученному результату. При этом возможно наложение эффектов ошибок. Кроме того, ошибка в одном модуле может блокировать тестирование другого.
- **Трудно организовать исправление ошибок.** В результате тестирования тестировщиком фиксируется найденная проблема. Дефект в системе, вызвавший эту проблему, будет устранять разработчик. Поскольку, как правило, тестируемые модули написаны разными людьми, возникает проблема - кто из них является ответственным за поиск устранение дефекта? При такой "коллективной безответственности" скорость устранения дефектов может резко упасть.
- **Процесс тестирования плохо автоматизируется**. Преимущество (нет дополнительного программного обеспечения, сопровождающего процесс тестирования) оборачивается недостатком. Каждое внесённое изменение требует повторения всех тестов.

Инкрементальный метод

Характеризуется пошаговым (помодульным) наращиванием комплекса программ с пошаговым тестированием собираемого комплекса.

В инкрементальном методе выделяют две стратегии добавления модулей:

- "**Сверху вниз**" и соответствующее ему **нисходящее тестирование**.
- "**Снизу вверх**" и соответственно ему **восходящее тестирование**.

Нисходящее тестирование

- Предполагает, что процесс интеграционного тестирования движется следом за разработкой.

Сначала тестируют только самый верхний управляющий уровень системы, без модулей более низкого уровня.

Затем постепенно с более высокоуровневыми модулями интегрируются более низкоуровневые.

В результате применения такого метода отпадает необходимость в драйверах (роль драйвера выполняет более высокоуровневый модуль системы), однако сохраняется нужда в заглушках. По мере тестирования заглушки по очереди заменяются на реальные модули.

Восходящее тестирование

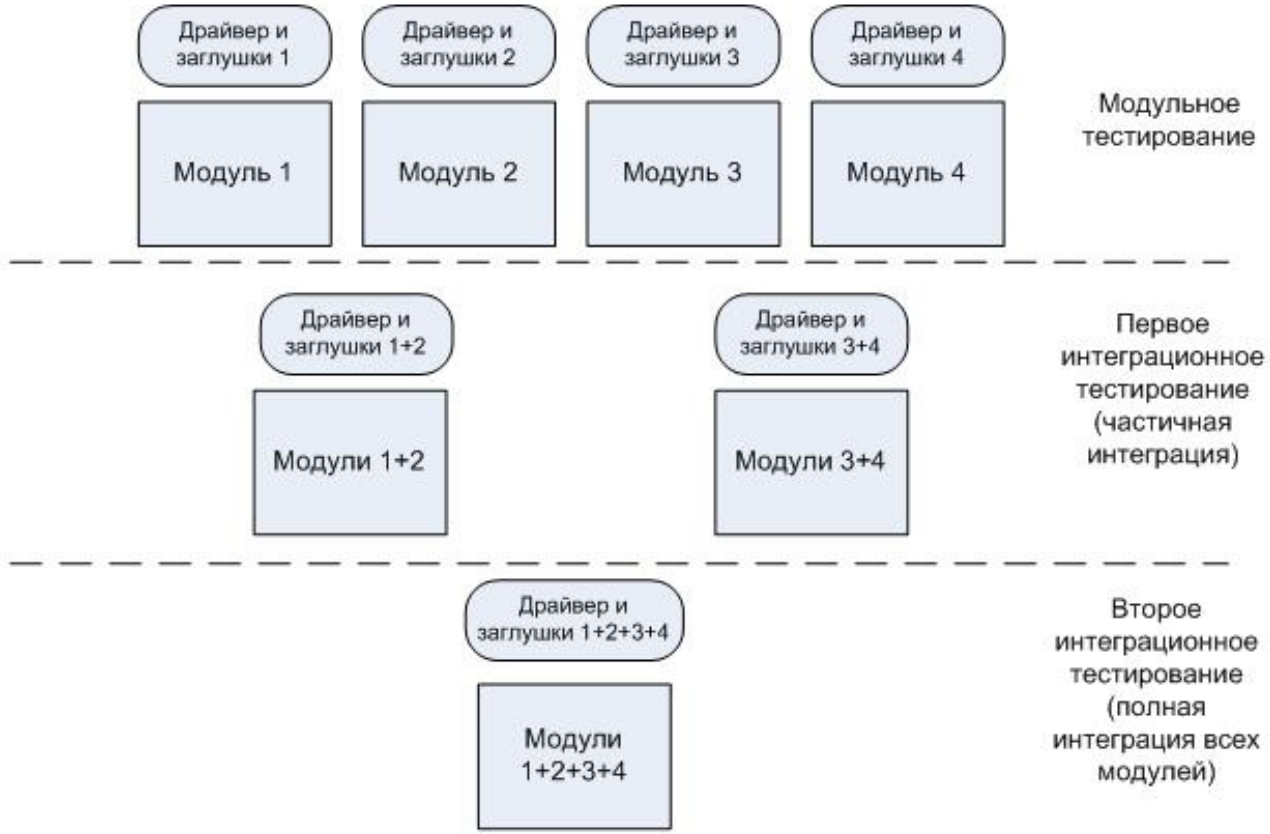
При использовании этого метода подразумевается, что сначала тестируются все программные модули, входящие в состав системы и только затем они объединяются для интеграционного тестирования.

При таком подходе значительно упрощается локализация ошибок: если модули протестированы по отдельности, то ошибка при их совместной работе есть проблема их интерфейса.

При таком подходе область поиска проблем у тестировщика достаточно узка, и поэтому гораздо выше вероятность правильно идентифицировать дефект.

Восходящее тестирование

Однако, у восходящего метода тестирования есть существенный недостаток - необходимость в разработке драйвера и заглушек для модульного тестирования перед проведением интеграционного тестирования и необходимость в разработке драйвера и заглушек при интеграционном тестировании части модулей системы



Классификация методов интеграционного тестирования

На практике чаще всего в различных частях проекта применяются все рассмотренные методы в совокупности.

Каждый модуль тестируют по мере готовности отдельно, а потом включают в уже готовую композицию.

Для одних частей тестирование получается нисходящим, для других - восходящим.

В рамках классификации по времени интеграции выделяют:

- тестирование с поздней интеграцией
- тестирование с постоянной интеграцией;
- тестирование с регулярной или послойной интеграцией.

Тестирование с поздней интеграцией

Практически полный аналог монолитного тестирования.

Интеграционное тестирование при такой схеме откладывается на как можно более поздние сроки проекта. Этот подход оправдан в том случае, если система является конгломератом слабо связанных между собой модулей, которые взаимодействуют по какому-либо стандартному интерфейсу, определенному вне проекта (например, в случае, если система состоит из отдельных Web-сервисов).

Схематично тестирование с поздней интеграцией может быть изображено в виде цепочки

R-C-V-R-C-V-R-C-V-I-R-C-V-R-C-V-I, где

- **R** - разработка требований на отдельный модуль,
- **C** - разработка программного кода,
- **V** - тестирование модуля,
- **I** - интеграционное тестирование всего, что было сделано раньше.

Тестирование с постоянной интеграцией

Подразумевает, что, как только разрабатывается новый модуль системы, он сразу же интегрируется со всей остальной системой.

При этом тесты для этого модуля проверяют как сугубо его внутреннюю функциональность, так и его взаимодействие с остальными модулями системы.

Таким образом, этот подход совмещает в себе модульное тестирование и интеграционное. Разработки заглушек при таком подходе не требуется, но может потребоваться разработка драйверов.

Обычно именно этот подход называют *unit testing*, несмотря на то, что в отличие от классического модульного тестирования здесь не проверяется функциональность изолированного модуля.

Тестирование с постоянной интеграцией

Схематично тестирование с поздней интеграцией может быть изображено в виде цепочки

R-C-I-R-C-I-R-C-I, где

- **R** - разработка требований на отдельный модуль,
- **C** - разработка программного кода,
- **I** - интеграционное тестирование всего, что было сделано раньше.

Фаза тестирования модуля намеренно опущена и заменена на тестирование интеграции.

Тестирование с регулярной или послойной интеграцией

При данном тестировании интеграционному тестированию подлежат сильно связанные между собой группы модулей (слои), которые затем также интегрируются между собой.

Такой вид интеграционного тестирования называют также **иерархическим интеграционным тестированием**, поскольку укрупнение интегрированных частей системы, как правило, происходит по иерархическому принципу.

Однако, в отличие от нисходящего или восходящего тестирования, направление прохода по иерархии в этом подходе не задано.

Планирование интеграционного тестирования

На этапе планирования разрабатывается концепция и стратегия интеграции - документ, где описан общий подход к определению последовательности, в которой должны интегрироваться модули.

Составляется интеграционный тест-план (обычно кластерного типа), в котором для каждого кластера из интегрированных модулей определяется следующее:

- ✓ кластеры, от которых зависит данный кластер;
- ✓ кластеры, которые должны быть протестированы до тестирования данного кластера;
- ✓ описание функциональности тестируемого кластера;
- ✓ список модулей в кластере;
- ✓ описание тестовых примеров для проверки кластера.

Планирование интеграционного тестирования

Планирование интеграционного тестирования должно быть синхронизировано с общим планом проекта, причем выделяемые для интеграционного тестирования модули и сроки их тестирования должны учитывать приоритеты важности частей системы. Зачастую рассмотрение приоритетов связано с тем, что системы разрабатываются в несколько этапов, на каждом из которых в эксплуатацию вводится только часть новой системы.

Интеграционное тестирование в данном случае должно укладываться в общий план-график проекта и учитывать затраты ресурсов на тестирование интеграции с уже работающими частями системы.

Системное тестирование

Системное тестирование.

- По завершению интеграционного тестирования все модули системы являются согласованными по интерфейсам и функциональности.

Переходят к системному тестированию.

Для системного тестирования применяется подход черного ящика, при этом в качестве входных и выходных данных используются реальные данные, с которыми работает система, или данные, подобные им.

Системное тестирование - один из самых сложных видов тестирования.

Системное тестирование. Задачи

Выявление дефектов, связанных с работой системы в целом:

- неверное использование ресурсов системы
- непредусмотренные комбинации данных пользовательского уровня
- несовместимость с окружением
- непредусмотренные сценарии использования
- отсутствующая или неверная функциональность
- неудобство в применении и тому подобное.

Категории тестов системного тестирования

1. Полнота решения функциональных задач.
2. Тестирование целостности (соответствие документации, комплектность).
3. Стрессовое тестирование - на предельных объемах нагрузки входного потока.
4. Корректность использования ресурсов (утечка памяти, возврат ресурсов).
5. Оценка производительности.
6. Эффективность защиты от искажения данных и некорректных действий.
7. Проверка инсталляции и конфигурации на разных платформах.
8. Корректность документации

Функциональное тестирование

Данный вид тестирования предназначен для доказательства того, что вся система в целом ведет себя в соответствии с ожиданиями пользователя, формализованными в виде системных требований.

В ходе данного вида тестирования проверяются все функции системы с точки зрения ее пользователей (как пользователей-людей, так и "пользователей" - других программных систем).

Система при функциональном тестировании рассматривается как черный ящик, поэтому в данном случае полезно использовать классы эквивалентности.

Функциональное тестирование. Критерии полноты тестирования

Критерием полноты тестирования в данном случае будет полнота покрытия тестами системных функциональных требований (или системных тест-требований) и полнота тестирования классов эквивалентности, а именно:

- все функциональные требования должны быть протестированы;
- все классы допустимых входных данных должны корректно обрабатываться системой;
- все классы недопустимых входных данных должны быть отброшены системой, при этом не должна нарушаться стабильность ее работы;
- в тестовых примерах должны генерироваться все возможные классы выходных данных системы;
- во время тестирования система должна побывать во всех своих внутренних состояниях, пройдя при этом по всем возможным переходам между состояниями.

Тестирование производительности

Данный вид тестирования направлен на определение того, что система обеспечивает должный уровень производительности при обработке пользовательских запросов.

Тестирование производительности выполняется при различных уровнях нагрузки на систему, на различных конфигурациях оборудования.

Тестирование производительности позволяет выявлять узкие места в системе, которые проявляются в условиях повышенной нагрузки или нехватки системных ресурсов. В этом случае по результатам тестирования проводится доработка системы, изменяются алгоритмы выделения и распределения ресурсов системы.

Стрессовое тестирование.

Стрессовое тестирование имеет много общего с тестированием производительности, однако его основная задача - не определить производительность системы, а оценить производительность и устойчивость системы в случае, когда для своей работы она выделяет максимально доступное количество ресурсов либо когда она работает в условиях их критической нехватки.

Основная цель стрессового тестирования - вывести систему из строя, определить те условия, при которых она не сможет далее нормально функционировать.

Тестирование конфигурации.

Большинство программных систем массового назначения предназначено для использования на самом разном оборудовании.

В ходе тестирования конфигурации проверяется, что программная система корректно работает на всем поддерживаемом аппаратном обеспечении и совместно с другими программными системами. Необходимо также проверять, что система продолжает стабильно работать при горячей замене любого поддерживаемого устройства на аналогичное. При этом система не должна давать сбоев ни в момент замены устройства, ни после начала работы с новым устройством.

Тестирование безопасности.

Если программная система предназначена для хранения или обработки данных, содержимое которых представляет собой тайну определенного рода (личную, коммерческую, государственную и т.п.), то к свойствам системы, обеспечивающим сохранение этой тайны, будут предъявляться повышенные требования. Эти требования должны быть проверены при тестировании безопасности системы.

В ходе этого тестирования проверяется, что информация не теряется, не повреждается, ее невозможно подменить, а также к ней невозможно получить несанкционированный доступ, в том числе при помощи использования уязвимостей в самой программной системе.

Тестирование надежности и восстановления.

Если программная система предназначена для хранения или обработки данных, содержимое которых представляет собой тайну определенного рода (личную, коммерческую, государственную и т.п.), то к свойствам системы, обеспечивающим сохранение этой тайны, будут предъявляться повышенные требования.

Эти требования должны быть проверены при тестировании безопасности системы.

Тестирование надежности и восстановления.

Для корректной работы системы в любой ситуации необходимо удостовериться в том, что она восстанавливает свою функциональность и продолжает корректно работать после любой проблемы, прервавшей ее работу.

При тестировании восстановления после сбоев имитируются сбои оборудования или окружающего программного обеспечения либо сбои программной системы, вызванные внешними факторами.

После системного тестирования…

При разработке массового ПО после проведения системного тестирования система проходит этапы альфа- и бета-тестирования, во время которого работу системы проверяют потенциальные пользователи (либо специально выделенные фокус-группы пользователей, либо все желающие).

На этом этапе в программную систему вносятся последние незначительные изменения, не влияющие на суть системы.

После завершения этой стадии система поступает в продажу конечным пользователям.

После системного тестирования…

- При разработке заказного программного обеспечения фазу альфа- и бета-тестирования заменяют приемо-сдаточные испытания.

Во время этих испытаний заказчик удостоверяется, что система работает в соответствии с его потребностями (как зафиксированными в техническом задании на систему, так и не зафиксированными).

Заказчик может проводить такие испытания самостоятельно, выполняя заранее подготовленные тесты системы, либо проводить их совместно с представителями коллектива разработчиков. В этом случае тестовые примеры также готовятся разработчиками, например, на основе тестовых примеров, использовавшихся на этапе системного тестирования.

Сертификационные испытания.

После проведения системного тестирования и приемо-сдаточных испытаний проводятся сертификационные испытания.

Сертификация программного обеспечения - процесс установления и официального признания того, что разработка ПО проводилась в соответствии с определенными требованиями.

Контрольные вопросы:

1.Интеграционное тестирование. Цель.

2. Что представляют монолитный и инкрементальный методы сборки модулей.

3. Описать восходящее и нисходящее тестирование.

4. Системное тестирование. Задачи.

5. Категории тестов системного тестирования.