1. Типы интеграции и интеграционного взаимодействия.

Интеграция программных систем и продуктов — это обмен данными между системами с возможной последующей их обработкой. Смысл интеграции заключается в том, чтобы данные, которые пользователь вводит в одну систему, автоматически переносились в другую. Продукт, в который пользователь вводит данные, называется источник.

Например, если вы объединяете конфигурацию 1С: Торговля с 1С: Бухгалтерией, вам может потребоваться передать данные по всем продажам в бухгалтерию, а обратно получить сведения об оплате по этим продажам.

- Процесс интеграции делится на такие этапы:
- Определяем, какой продукт является источником, какой – приемником.
- Сопоставляем объекты между источником и приемником.
- Выбираем протокол для интеграции
- Проводим постобработку данных (после переноса в одну из сторон)

Web-интеграция представляет собой синхронизированное функционирование сайта, склада, бухгалтерии и других специализированных программ. Речь идет о бизнес-взаимодействии корпоративных ресурсов или интернет-магазинов с локальными информационными системами и любыми торгово-учетными решениями, при котором изменения в одном звене общей системы влияют на другие.

Также следует заметить, что web-интеграция актуальна не только для синхронизации систем eCommerce-бизнесов, но и для вывода любых бизнес-процессов в онлайн:

Интернет-интеграция позволяет развертывать информационные системы на базе сторонних приложений без необходимости разбираться в их системах.

Представляется возможность конструктировать комплексную функциональность, комбинируя разнородные компоненты посредством веб-протоколов или устанавливая приложения, представленные решением для интеграции или созданные сторонними разработчиками, в веб-интерфейсе.

Предоставляется доступ к веб-сервисам разработчиков.

Веб-сервисы используют программный язык и платформонезависимые интерфейсы между приложениями корпоративной инфраструктуры ИТ, что дает очевидные преимущества в поддержке, управляемости и развертывании информационных сетей.

Типы интеграции.

- Интеграция на уровне представления. Уровень представления — веб-базированный пользовательский интерфейс, платформозависимый графический пользовательский интерфейс (GUI) или консоль терминала. Данный уровень позволяет пользователю взаимодействовать с приложением. Интеграция на уровне представления даёт доступ к пользовательскому интерфейсу удаленных приложений. **Тестирование включает в себя следующие этапы:** составление тест-плана, создание тест-кейсов и юз-кейсов, выполнение тестов после создания пользовательского интерфейса, выявление ошибок, повторное тестирование после их исправления, подготовка отчета о тестировании.
- Интеграция на уровне функциональности. Данная интеграция подразумевает обеспечение прямого доступа к бизнес-логике приложений. Это достигается непосредственным взаимодействием приложений с API (программному интерфейсу приложений) или же взаимодействием посредством веб-сервисов. **Тестирование включает в себя следующие этапы:** составление тест-плана, создание тест-кейсов и юз-кейсов, выполнение тестов после интеграции по API, выявление ошибок, повторное тестирование после их исправления, подготовка отчета о тестировании.
- Интеграция на уровне данных. В данном случае предполагается доступ к одной или нескольким базам данных, используемых удаленным приложением. **Тестирование включает в себя следующие этапы:** составление тест-плана, создание тест-кейсов и юз-кейсов, выполнение тестов баз данных, выявление ошибок, повторное тестирование после их исправления, подготовка отчета о тестировании.
- Комплексная интеграция. Коммерческие решения по веб-интеграции, как правило, включают все три типа интеграции. **Тестирование включает в себя следующие этапы:** составление стратегии тестирования, тест-плана, создание тест-кейсов и юз-кейсов, выполнение тестов после проведения всех этапов комплексной интеграции: создание и подготовки базы данных, подготовка пользовательского интерфейса, проведение

пользовательского тестирования, тестирования функциональности приложения/сервиса, выявление ошибок, повторное тестирование после их исправления, подготовка отчета о тестировании.

Также существует следующее **деление по типам интеграции**:

1. Внутренняя и внешняя.

Например, интеграция Тильды (конструктор, на котором сделан сайт) и Sendpulse (приложение для рассылок писем). Это две разные системы, которые разработали две разные компании, это внешняя интеграция. Чтобы показывать карту на сайте требуется внешняя интеграция (со сторонним сервисом - GoogleMap).

Внутренняя интеграция. Например, система технического учёта и система CRM могут быть интегрированы между собой в Компании. Создание единой IT-инфраструктуры предполагает интеграцию приложений и данных, с помощью которых автоматизируется управление различными процессами. Интеграция не вызывает особых сложностей, когда речь идет о 2 информационных системах. Однако со временем IT-инфраструктура предприятия может усложняться, и интеграция уже трех и более автоматизированных систем, как и их совместная работа, может вызывать определенные трудности.

В результате изменения процессов и внедрения новых информационных систем и технологий, IT-инфраструктура практически каждой компании становится малоэффективной: количество связей между различными автоматизированными системами и их сложность растет в геометрической прогрессии, возникает двойной ввод информации, разночтения в нормативно-справочной информации и многие другие проблемы, приводящие, в конечном счете, к замедлению процессов и ошибкам в оперативных, тактических и стратегических решениях. В итоге появляется жизненная необходимость интегрировать приложения путем создания единой инфраструктуры, в которой все процессы и все информационные потоки связаны друг с другом и протекают быстро и без сбоев.

2. Простая и сложная. Интеграция может быть с использованием no-code инструментов (например, Integromat), а может требовать разработчиков и написания кода. Можно выделить еще ручную интеграцию. Например, сотрудник склада, переносящий данные из системы 1С:Склад в CRM, занимается интеграцией этих систем.

3. Классификация по целям.

Интеграция может иметь разные цели. В основном, речь идёт про организацию сквозного бизнес-процесса (процесс заказа продуктов, например, требует работы нескольких систем). Или про организацию хранения данных (MDM-класс систем, DWH-системы и т.п.). Или про представление данных в одном окне (например, оператор мог видеть dashboard с кучей информации из разных систем): всё это ради экономии времени, денег или предоставлении какой-то новой, ранее невозможной услуги (сервиса) или продукта.

4. Интеграция может быть стихийной или плановой.

Вам срочно надо объединить две системы, чтобы организовать процесс. ИЛИ ещё систему CRM к ним интегрировать. И так год за годом. Куча систем, как клубок, наматывается. Плановая, соответственно, возникает, когда есть план.

5. Интеграции могут отличаться по структуре связи.

Взаимодействие систем по принципу точка-точка исторически появилось раньше остальных. Подразумевает этот принцип всего лишь ситуацию, когда каждая система попарно интегрируется с другими.

А интегрируется с Б, Б с В и др., т.е. напрямую. Система А знает, как обратиться к системам Б и В, завязана на их интерфейсы, знает адрес сервера приложений.

Звезда — это когда мы добавляем, к примеру, шину (ESB). Какую-то прослойку, которая позволяет всем системам интегрироваться между собой. Это как единая точка. Шина — это тоже ПО. Звезда имеет много плюсов (например, если эта CRM система вам больше не нужна, меняем её на новую — всё не развалится). Но есть и минусы. Например, у вас есть единая точка отказа. Случись что-то с шиной — и не работает вообще всё (весь контур систем, которые интегрируются через эту шину).

Смешанная — это когда у нас на проекте есть шина, общая точка, где системы

объединяются. Но и есть попарно интегрированные системы.

6. Синхронная или асинхронная интеграция.

Асинхронная интеграция нужна — это как сообщение в почте или чате, ответа мы не ждём сразу. Синхронная — это как общение по телефону: если ты звонишь человеку, то ты общаешься с ним и ждёшь ответа сразу во время разговора.

7. Интеграция систем, подсистем, сервисов, микросервисов.

Если вы интегрируете свою систему CRM с чужой системой MDM, то это интеграция систем. Если сервису Яндекс.Кошелек вдруг нужны данные с сервиса Госуслуги, то перед нами интеграция сервисов. Если наша система построена по SOA, то мы имеем распределенную систему, подсистемы которой интегрируются между собой (например, с помощью шины, о которой будет ниже). Для систем на микросервисах все тоже самое.

8. Паттерны интеграции по типу обмена данными.

Речь идет о таких паттернах, как обмен файлами, обмен через общую базу данных, удаленный вызов процедур, обмен сообщениями.

Самая простая (и исторически возникшая самой первой) — это интеграция по паттерну «общая база данных». Быстро, дешево, небезопасно. Не используйте этот тип интеграции, если у вас есть хоть какая-то альтернатива. В этом подходе несколько информационных систем используют одну общую логическую структуру данных. Примеры технологий: Oracle, MS SQL Server, SAP HANA, MongoDB.

«Обмен файлами» — это простой способ передать файлы. Выкладываете отчет на FTP-сервер как на гугл-диск, а потом другая система его скачивает. Дешево, достаточно безопасно, но сложного взаимодействия не построить (попробуйте впятером работать с одной страницей в гугл-док, не общаясь другими каналами связи.

Исторически это самый ранний подход к интеграции, который отличается относительной простотой. Суть его в следующем: одно приложение создаёт файл, а другое приложение считывает данный файл. Приложения интегрирующиеся данным методом должны договориться об подходе к именованию файлов, их расположении, формате и процедуре удаления. Яркими примерами технологий, использующихся в данном типе интеграции, является FTP(s), NFS, HDFS (Hadoop), AWS S3 и IBM MQ File Transfer Edition.

На основе «удалённый вызов процедур (RPC)» можно построить куда больше взаимодействий. Если вам нужно построить интеграцию между двумя системами, то скорее всего это REST — вот и все, что стоит пока запомнить. Одно приложение предоставляет доступ к своей функциональности с помощью удалённого вызова процедуры. Примеры технологий: SOAP, Java RMI, .NET Remoting, JSON-RPC, XML-RPC, gRPC, gRPC, REST, SOAP, GraphQL (формально не все из этого RPC).

Паттерн «обмен сообщениями» становится необходим в одном из таких случаев, например:

✓ крупный проект, в котором мы хотим иметь единую точку для управления интеграциями

✓ асинхронном взаимодействии

✓ больших потоках информации

✓ масштабировании

Вот тогда появляется сервисная шина предприятия или брокер. Это дорого, долго, но порой необходимо.

9. Паттерны по методу интеграции:

• Интеграция систем по данным (data-centric).

• Объектно-центрический (object-centric).

• Функционально-центрический (function-centric) подход.

• Интеграция на основе единой понятийной модели предметной области (concept-centric).

Аналитик не выбирает паттерн интеграции (это задача архитектора), но ориентироваться в них весьма желательно.

10. Системная интеграция

Многие организации используют устаревшее программное обеспечение для выполнения своих основных бизнес-функций. Ее нельзя удалить и заменить более современной технологией, поскольку она имеет решающее значение для повседневного рабочего процесса компании. Вместо этого унаследованные системы можно модернизировать , установив канал связи с более новыми информационными системами и технологическими решениями.

Пример: подключение устаревшей CRM-системы к хранилищу данных или системе управления перевозками (TMS) .

Интеграция корпоративных приложений (EAI)

Цель: объединение разных подсистем внутри одной бизнес-среды

По мере своего роста компании внедряют все больше и больше корпоративных приложений для оптимизации своих фронт- и бэк-офисных процессов. Эти приложения часто не имеют общих точек соприкосновения и накапливают огромные объемы данных по отдельности. Интеграция корпоративных приложений (EAI) объединяет все функции в одну бизнес-цепочку и автоматизирует обмен данными в режиме реального времени между различными приложениями.

Пример: создание одной экосистемы для бухгалтерского учета, информации о человеческих ресурсах, управления запасами, планирования ресурсов предприятия (ERP) и CRM-систем компании.

Интеграция со сторонними системами

Цель: расширение функционала существующей системы

Интеграция сторонних инструментов — отличный вариант, когда вашему бизнесу нужны новые функции, но вы не можете позволить себе разработку программного обеспечения на заказ или у вас просто нет времени ждать, пока функции будут созданы с нуля.

Пример: интеграция существующего приложения с системами онлайн-платежей (PayPal, WebMoney), социальными сетями (Facebook, LinkedIn), сервисами потокового онлайн-видео (YouTube) и т. д.

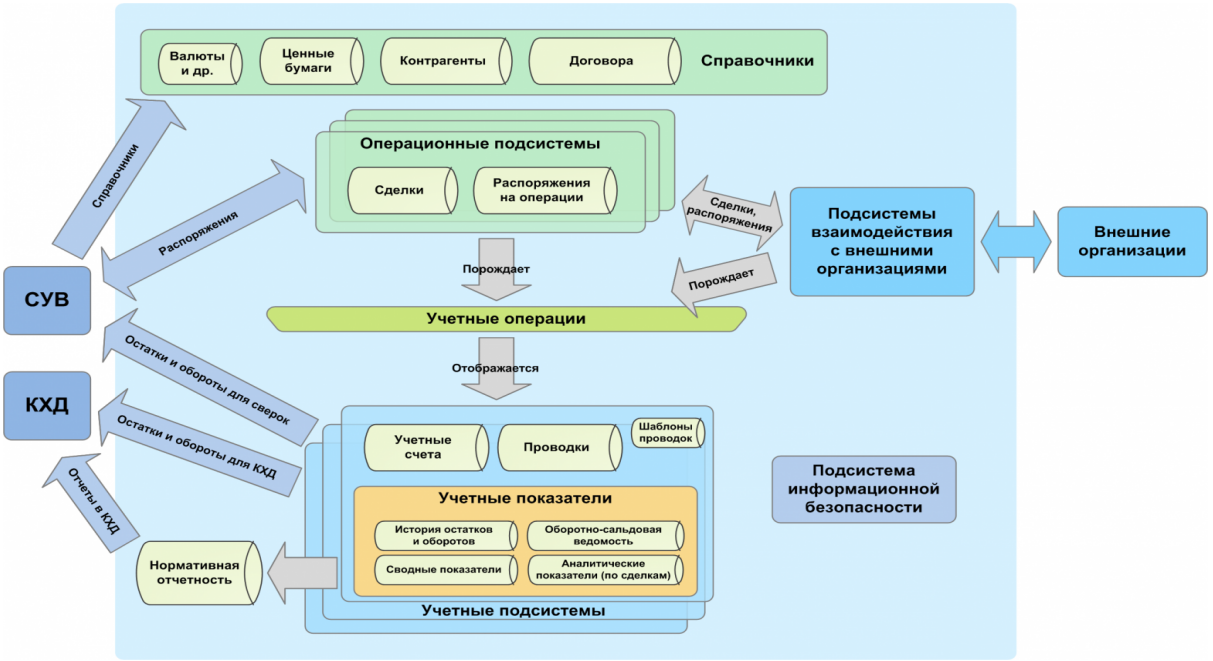
Интеграция между компаниями

Цель: соединение систем двух и более организаций

Интеграция B2B или Business-to-Business автоматизирует транзакции и обмен документами между компаниями. Это приводит к более эффективному сотрудничеству и торговле с поставщиками, клиентами и партнерами.

Пример: подключение системы закупок розничного продавца к ERP-системе поставщика.

В любой ситуации основная цель системной интеграции всегда одна и та же — собрать воедино разрозненные и разделенные части посредством построения целостной сети. Давайте посмотрим на существующие технологии и архитектурные модели, которые делают возможным волшебство интеграции.



11.Достаточно часто данные переносят в обе стороны, например, после преобразования в системе-приемнике результаты отправляются обратно в источник. А потому интеграция бывает как односторонней, так и двухсторонней. При двухстороннем обмене необходимо хранить уникальный идентификатор для всех систем, которые участвуют в интеграции. И я считаю, что его также стоит дублировать в обеих системах.

2. Процесс тестирования интеграционного взаимодействия.

Любая интеграция - это часть системы и определенный функционал, тестируем как отдельно каждую интеграцию с помощью интеграционного тестирования, так и в процессе тестирования приложения/сервиса вцелом, возможно использование следующих видов тестирования:

интеграционное, исследовательское, совместимости, конфигурационное, надежности и восстановления, производительности, удобство использования, комплексное, входной тест, основное, регрессионное, приемочное, динамическое, системное, нагрузочное, безопасности, функциональное.

Интеграционное тестирование — это всего лишь вторая фаза полного цикла тестирования, которая следует за модульным тестированием. При этом важность этого этапа сложно переоценить, так как на этом этапе тестируются взаимодействия между модулями мобильного или веб-приложения. Общая производительность программного обеспечения будет зависеть от того, как отдельные модули работают вместе. Вот почему так важно доверить интеграционное тестирование команде профессионалов.

Тестирование интеграции каждого из типов тестирования включает в себя:

1. Проверку интеграции с админкой. Админка определяет, насколько быстро и технологично служба поддержки может разобраться с возникшими инцидентами и устранить проблему.

При тестировании очередное неудачное обновление с ошибкой запросто может породить вал из сотен или тысяч ошибок. Все таки тестовый контур не полноценно эмулирует реальный поток данных.

2. Проверку при обработке сообщений. Прием сообщений не должен полностью останавливаться на первой же ошибке. Остановка потока при любой ошибке — плохо, так как при любой частной ошибке, вызванной, например, обработкой нового типа сообщений, мы получаем вал инцидентов и нарушение взаимодействия систем.

3. Проверку последовательности обработки сообщений. Для конкретной цепочки нарушение порядка часто чревато ошибками на системе-приемнике.

4. Возможность посмотреть сообщение, письмо или вызов, который будет выполняться. Важно показать не только конверт сообщения, но и внутреннее его содержание в структурном виде, несмотря на обобщенный характер таких интерфейсов при тестировании. Длинный упакованный xml или json — не слишком хороший вариант.

5. Умение запустить сообщение на обработку в системе-получателе. Это актуально, если при обработке была внутренняя ошибка в коде или данных, которую исправили. Это часто бывает при расширении протокола интеграции — новые поля обрабатываются неверно, ошибку исправляют, и после этого для исправления данных необходимо повторно обработать сообщения. Аналогично может быть нужно, если сообщения оказались чувствительны к порядку обработки, а это не предусмотрели.

6. Умение найти сообщение в системе-источнике и повторно его отправить. Иногда сообщения теряются по дороге. Протокол http не гарантирует ни доставку сообщений, ни получение ответа. При этом повторная отправка при отсутствии ответа чревата дублированием сообщений — ведь потерять могли именно ответ, а не само сообщение.

7. Видеть на интерфейсе те изменения, которые сообщение вносит в существующий объект. Идеально, когда мы можем видеть также изменения предыдущих сообщений (которые его меняли) и последующих необработанных (при их наличии).

В случае проблем полезна возможность отправить из системы-отправителя объект целиком, со всеми его атрибутами. И, конечно, принять его, проигнорировав при этом предыдущие ошибочные сообщения.

Хороший инструмент — возможность протестировать и изменить объект вручную средствами системы — возможно, с выходом в нештатный режим для объектов, правки которых запрещены, и с последующим контролем, что объект действительно получился идентичным.

8. Поддержку распределенных ограничений целостности. Бизнес-логика очень часто накладывает ограничения на атрибуты объектов не только в пределах одного объекта, но и в совокупности. Например, НДС в заказе зависит от категории товара и от налогового режима контрагента — покупателя или продавца. И такие ограничения важно контролировать, особенно если ошибки ведут к налоговым или другим регуляторным последствиям. В случае, если

изменения происходят в рамках одной системы, тестировать относительно просто: при создании заказа проверяются атрибуты товара и контрагента, а при изменении атрибутов контрагентов или товаров — что это изменение не нарушает существующих данных.

9. Сверку данных. Интеграция должна включать систему независимой сверки данных между системами в автоматическом режиме или по запросу. И хорошо, если это поддерживает админка, а не разработчик, вручную выгружающий данные из систем и сравнивающий их. Очень много ошибок, как показывает практика, связано с эффектами косвенного изменения объектов, которые не попадают в сообщения по интеграции. Например, когда изменения повешены на триггеры базы данных, и сервер приложений о них не в курсе. Поэтому важно при сверке передавать интегральные показатели, причем отражающие не только то, что попало в поток интеграции, а характеризующие множества синхронизируемых объектов в целом.

10. Отслеживание ошибок и дефектов возможно постоянно. Интеграция – процесс сложный, и проблемы из-за человеческого фактора возникают достаточно часто, защититься от них практически не реально. Также бывают и программные сбои, особенно это касается таких сложных систем с большим числом багов, как программные продукты 1С. А для бизнеса очень важно, чтобы обмен данными проходил своевременно, а если возникла проблема также важно ее оперативно устранить.

Самые распространенные решения для отслеживания проблем при передачи данных в интеграции:

При помощи смс, электронного письма, всплывающих уведомлений о сбое должен получить человек, который занимается процессом, например, менеджер продаж..

Для контроля аналогичное уведомление (чаще всего на email) отправляется руководителю отдела или директору компании.

Обязательно ведется лог-файл ошибок для того, чтобы специалист смог просмотреть все подробности.

Если обмен данными не происходит, тестируем:

Ограничение доступа по IP.

Ограничение прав пользователя.

Ограничение по количеству обращений к источнику или приемнику.

Интеграционное тестирование – это тип тестирования, при котором программные модули объединяются логически и тестируются как группа. Как правило, программный продукт состоит из нескольких программных модулей, написанных разными программистами. Целью нашего тестирования является выявление багов при взаимодействии между этими программными модулями и в первую очередь направлен на проверку обмена данными между этими самими модулями. Именно поэтому оно также называется «I & T» (интеграция и тестирование), «тестирование строк» и иногда «тестирование потоков».

Каждый программный модуль проходит отдельные этапы тестирования (модульное тестирование), но не смотря на это, дефекты могут оставаться по ряду причин:

- Поскольку, как правило, модули разрабатываются разными специалистами, их понимание и логика программирования могут отличаться. Тут интеграционное тестирование становится необходимым для проверки взаимодействия модулей между собой.
- Во время разработки модуля заказчики часто меняют требования, и если у вас сжатые сроки требования могут попросту не успеть пройти модульное тестирование, и, следовательно, системная интеграция может пройти с помехами. Опять получается, что от интеграционного тестирования не убежать.
- Интерфейсы программных модулей с базой данных могут быть ошибочными
- Внешние аппаратные интерфейсы, если таковые имеются, могут быть ошибочными
- Неправильная обработка исключений может вызвать проблемы.

Интеграционное тестирование отличается от других видов тестирования тем, что он сосредоточен в основном на интерфейсах и потоке данных (между модулями). Здесь приоритет проверки присваивается интегрирующим ссылкам, а не функциям блока, которые уже проверены.

Пример тестирования интеграции для следующего сценария:
Приложение имеет 3 модуля, например «Страница входа», «Почтовый ящик» и «Удалить электронную почту». Каждый из них интегрирован логически.

Здесь нет нужды тестировать страницу входа, т.к. это уже было сделано в модульном тестировании. Но проверьте, как это интегрировано со страницей почтового ящика.

Аналогично, «Почтовый ящик»: проверьте его интеграцию с модулем «Удалить электронную почту».

Идентификатор теста	Цель теста	Описание теста	Ожидаемый результат
1	Проверьте интерфейсную связь между модулем входа в систему и почтовым ящиком.	Введите учетные данные и нажмите кнопку «Войти»	Быть направленным в почтовый ящик
2	Проверьте интерфейсную ссылку между почтовым ящиком и модулем удаления почты.	Из почтового ящика выберите адрес электронной почты и нажмите кнопку удаления	Выбранное письмо должно появиться в папке «Удаленные / Корзина»

Стратегии, методологии и подходы в интеграционном тестировании.

Программная инженерия задает различные стратегии интеграционного тестирования:
- Подход Большого взрыва.
- Инкрементальный подход:
    - Нисходящий подход (сверху вниз)
    - Подход «снизу вверх»
    - Сэндвич – комбинация «сверху вниз» и «снизу вверх»

Ниже приведены различные стратегии, способы их выполнения и их ограничения, а также преимущества.

Подход Большого взрыва
Все компоненты собираются вместе, а затем тестируются.
Преимущества:
- Удобно для небольших систем.

Недостатки:
- Сложно локализовать баги.
- Учитывая огромное количество интерфейсов, некоторые из них при тестировании можно запросто пропустить.
- Недостаток времени для группы тестирования, т.к тестирование интеграции может начаться только после того, как все модули спроектированы.
- Поскольку все модули тестируются одновременно, критические модули высокого риска не изолируются и тестируются в приоритетном порядке. Периферийные модули, которые имеют дело с пользовательскими интерфейсами, также не изолированы и не проверены на приоритет.

Инкрементальный подход.
В данном подходе тестирование выполняется путем объединения двух или более логически связанных модулей. Затем добавляются другие связанные модули и проверяются на правильность функционирования. Процесс продолжается до тех пор, пока все модули не будут соединены и успешно протестированы.

Поэтапный подход, в свою очередь, осуществляется двумя разными методами:

- Снизу вверх
- Сверху вниз

Заглушка и драйвер

Инкрементальный подход осуществляется с помощью фиктивных программ, называемых заглушками и драйверами. Заглушки и драйверы не реализуют всю логику программного модуля, а только моделируют обмен данными с вызывающим модулем.

Заглушка: вызывается тестируемым модулем.

Драйвер: вызывает модуль для тестирования.

Интеграция «снизу вверх»

В восходящей стратегии каждый модуль на более низких уровнях тестируется с модулями более высоких уровней, пока не будут протестированы все модули. Требуется помощь драйверов для тестирования.

Преимущества:

- Проще локализовать ошибки.
- Не тратится время на ожидание разработки всех модулей, в отличие от подхода Большого взрыва.

Недостатки:

- Критические модули (на верхнем уровне архитектуры программного обеспечения), которые контролируют поток приложения, тестируются последними и могут быть подвержены дефектам.
- Не возможно реализовать ранний прототип

Интеграция «сверху вниз»

При подходе «сверху вниз» тестирование, что логично, выполняется сверху вниз, следуя потоку управления программной системы. Используются заглушки для тестирования.

Преимущества:

- Проще локализовать баги.
- Возможность получить ранний прототип.
- Критические Модули тестируются на приоритет; основные недостатки дизайна могут быть найдены и исправлены в первую очередь.

Сэндвич (гибридная интеграция)

Эта стратегия представляет собой комбинацию подходов «сверху вниз» и «снизу вверх». Здесь верхнеуровневые модули тестируются с нижнеуровневыми, а нижнеуровневые модули интегрируются с верхнеуровневыми, соответственно, и тестируются. Эта стратегия использует и заглушки, и драйверы.

Как сделать интеграционное тестирование?

Алгоритм интеграционного тестирования:

1. Определение объема работ для выбора правильной стратегии интеграционных тестов. Главное, чтобы стратегия покрывала все приложение;
2. Тщательная проверка архитектуры программного обеспечения для определения критических модулей, требующих особого внимания;
3. Любые полученные тестовые данные не должны быть проигнорированы. Всегда необходимо сообщать о малейших проблемах для их дальнейшего решения;
4. Возможность общения с разработчиками программного обеспечения часто требуется при проведении интеграционного тестирования. Это помогает получить более точное представление об интерфейсах и особенностях приложения.
5. Подготовка план интеграционных тестов
6. Разработка тестовых сценариев.

7. Выполнение тестовых сценариев и фиксирование багов.
8. Отслеживание и повторное тестирование дефектов.
9. Повторять шаги 3 и 4 до успешного завершения интеграции.

Атрибуты Интеграционного тестирования.

Включает в себя следующие атрибуты:

- Методы / Подходы к тестированию (об этом говорили выше).
- Области применения и Тестирование интеграции.
- Роли и обязанности.
- Предварительные условия для Интеграционного тестирования.
- Тестовая среда.
- Планы по снижению рисков и автоматизации.

Критерии старта и окончания интеграционного тестирования.

Критерии входа и выхода на этап Интеграционного тестирования, независимо от модели разработки программного обеспечения

Критерии старта:

- Модули и модульные компоненты
- Все ошибки с высоким приоритетом исправлены и закрыты
- Все модули должны быть заполнены и успешно интегрированы.
- Наличие плана Интеграционного тестирования, тестовый набор, сценарии, которые должны быть задокументированы.
- Наличие необходимой тестовой среды

Критерии окончания:

- Успешное тестирование интегрированного приложения.
- Выполненные тестовые случаи задокументированы
- Все ошибки с высоким приоритетом исправлены и закрыты
- Технические документы должны быть представлены после выпуска Примечания.

Лучшие практики / рекомендации по интеграционному тестированию

- Сначала определите интеграционную тестовую стратегию, которая не будет противоречить вашим принципам разработки, а затем подготовьте тестовые сценарии и, соответственно, протестируйте данные.
- Изучите архитектуру приложения и определите критические модули. Не забудьте проверить их на приоритет.
- Получите проекты интерфейсов от команды разработки и создайте контрольные примеры для проверки всех интерфейсов в деталях. Интерфейс к базе данных / внешнему оборудованию / программному обеспечению должен быть детально протестирован.
- После тестовых случаев именно тестовые данные играют решающую роль.
- Всегда имейте подготовленные данные перед выполнением. Не выбирайте тестовые данные во время выполнения тестовых случаев.