# Integration_types

**Что такое интеграции между системами?**

Интеграция информационных систем — это процесс установки связей между информационными системами предприятий и организаций для получения единого информационного пространства и организации поддержки сквозных бизнес-процессов предприятий и организаций.

Интеграция систем в большинстве случаев – мера вынужденная, направленная на повышение эффективности бизнес-процессов компании, в которых используются информационные системы

**Типы интеграции — какие они бывают?**

1**. Нет интеграции между систем.**

Например, в компании используются три независимые информационные системы: «Складская система» (учет и анализ товародвижений на складе), «CRM-система» (учет и анализ продаж и других взаимоотношений с клиентами) и «Бухгалтерская система» (бухгалтерский учет и финансовый анализ). Между ними нет информационного обмена.

Это приводит к тому, что менеджеры по продажам после выставления счетов клиентам вынуждены печатать их копии и нести в бухгалтерию. В бухгалтерии они регистрируются в бухгалтерской системе. Бухгалтерия регистрирует поступление денег на счет. Менеджеры по продажам, не имея возможность получить оплаты автоматически в CRM-систему, вынуждены ежедневно осведомляться в бухгалтерии о поступлении денег от клиентов. В такой ситуации присутствует большой документооборот между менеджерами, бухгалтерией и складом, а также двойная регистрация действий (один раз в складской системе, второй раз в бухгалтерской).

Если посчитать затраты оплачиваемого компанией времени сотрудников на выполнение дублирующихся процедур в разных системах (выделены красным на схеме), то может получиться ощутимая доля в общих издержках фирмы.

**2. Простая и сложная** (очень условное название)

Интеграция может быть с использованием no-code инструментов, 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. Или про представление данных в одном окне: 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? Куча систем, как клубок, наматывается: то там интегрируем, то тут. Плановая, соответственно, возникает, когда есть план.

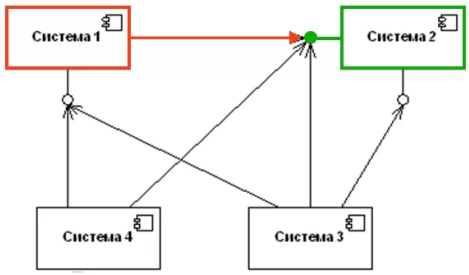
### 5. Интеграции могут отличаться по структуре связи

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.

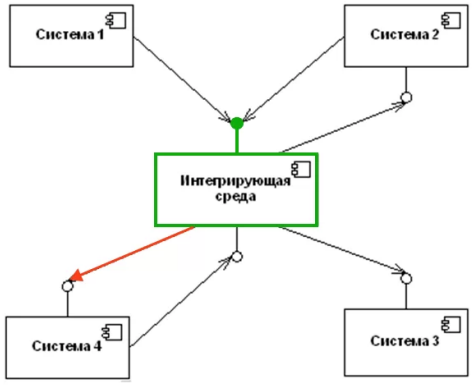
**Звезда —** это когда мы добавляем, к примеру, шину (ESB). Какую-то прослойку, которая позволяет всем системам интегрироваться между собой. Это как единая точка. Шина — это тоже ПО. Звезда имеет много плюсов (например, если эта CRM система вам больше не нужна, меняем её на новую — всё не развалится). 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.

Интеграция «Точка-точка»




Интеграция «Звезда»




**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.

Это большая тема, можно много говорить;

**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, удалённый 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.

**«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. На

основе **«удалённый вызов процедур (RPC)»** можно построить куда больше взаимодействий. Из всего многообразия сейчас максимально популярны gPRS, REST, SOAP, GraphQL. Если вам нужно построить интеграцию между двумя системами, то скорее всего это REST — вот и все, что стоит пока запомнить.

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

- крупный проект, в котором мы хотим иметь единую точку для управления интеграциями (помните п.5 «звезда»?)
- 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**

Никогда не встречала, чтобы кто-то всерьез пользовался этой классификацией, но так как она существует, поделюсь с вами:

• 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).

**Цели интеграции**

Общие цели интеграции приложений можно сформулировать следующим образом:

- уменьшить стоимость эксплуатации совокупности приложений предприятия;
- увеличить скорость выполнения типичных задач или гарантировать сроки их выполнения;
- поднять качество выполнения задач за счет формализации процессов и минимизации человеческого фактора, как основного источника ошибок.

В качестве целей конкретных интеграционных проектов обычно фигурируют более четкие формулировки. Но обычно все, в конце концов, сводится к общим

целям,
которые можно сформулировать в еще более общем виде — уменьшить операционные расходы предприятия или организации. Поэтому интеграционные
проекты часто оказываются в выигрышном положении с точки зрения обоснования перед людьми, принимающими решение о финансировании проектов: расчет показателей возврата инвестиций для таких проектов может выглядеть достаточно привлекательным.

Успешная интеграция корпоративных систем позволяет достичь и дополнительных целей — обеспечить автоматизированный контроль прохождения основных бизнес-процессов на предприятии, информационную безопасность при реализации бизнес-процессов и т.д.

**Что такое интеграционное тестирование?**

**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».

**Зачем нужно интеграционное тестирование?**


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.

**Примеры интеграционного тестирования**

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».

**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**

Здесь 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, так как 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.

**Недостатки:**

- Модули на более низком уровне тестируются неадекватно

**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. Подготовка план интеграционных тестов
2. Разработка тестовых сценариев.
3. Выполнение тестовых сценариев и фиксирование багов.

4. Отслеживание и повторное тестирование дефектов.
5. 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.