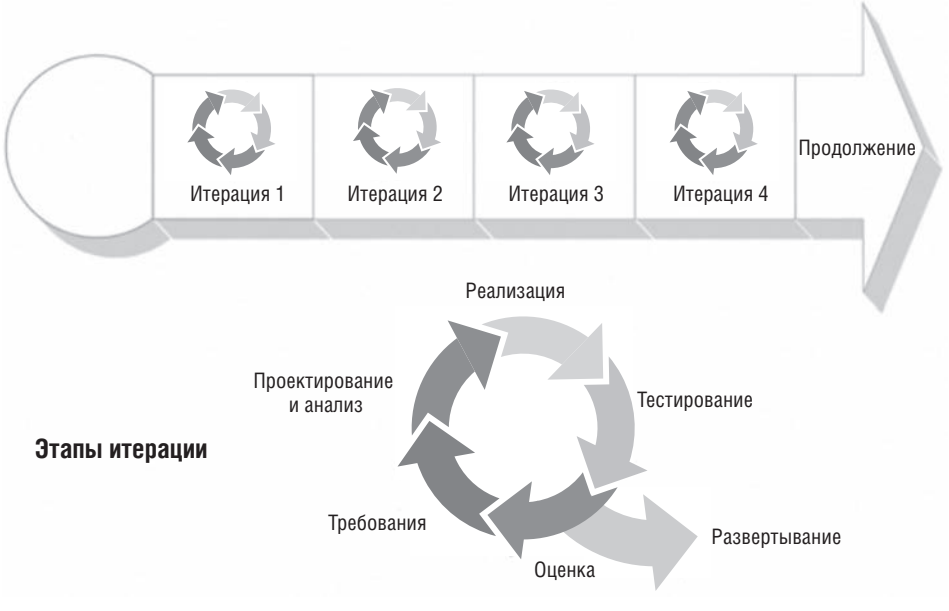
ИТЕРАТИВНЫЕ И ИНКРЕМЕНТНЫЕ МОДЕЛИ РАЗРАБОТКИ

Итеративная разработка — это процесс определения требований, проектирования, создания и тестирования системы, выполняемый в виде серии небольших инкрементных этапов. При итеративной разработке по мере прохождения итераций могут обнаруживаться новые требования или уточняться имеющиеся. Принцип такого подхода — «немного сделать, немного протестировать». Каждая итерация обеспечивает обратную связь для следующей итерации. По завершении итерации новые элементы необходимо протестировать совместно с существующей и не изменившейся частью продукта, поэтому регрессионное тестирование необходимо выполнять после каждой итерации.



ИТЕРАТИВНАЯ И ИНКРЕМЕНТНАЯ РАЗРАБОТКА

Примерами такой разработки являются:

Rational Unified Process, RUP (рациональный унифицированный процесс): каждая итерация, как правило, длится относительно долго (например, два-три месяца), и приращение функций на ней соответственно велико, например две или три группы связанных функций.

Scrum (agile-разработка): каждая итерация, как правило, длится относительно недолго (например, несколько дней или недель), и прирост функций на ней соответственно невелик, например несколько улучшений и/или две-три новые функции.

Kanban (agile-разработка): реализуется с итерациями фиксированной продолжительности или без них, при этом по завершении итерации может быть выпущено либо одно улучшение или функция, либо группа функций, выпускаемых одновременно.

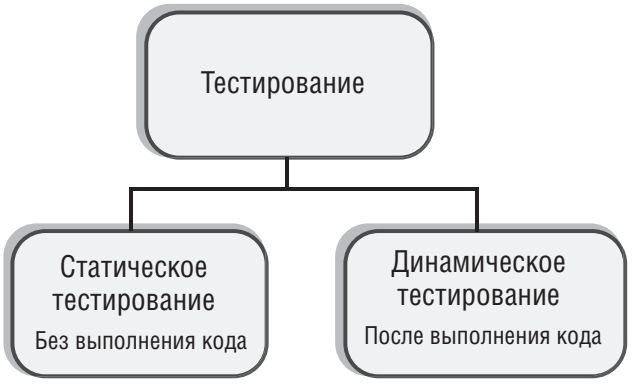
Спиральная модель разработки: предполагает создание экспериментальных приращений. Некоторые из этих приращений могут быть в дальнейшем переработаны или даже отброшены.

Приращение, полученное в результате итерации, может проходить тестирование на нескольких уровнях разработки. Приращение, добавленное к ранее разработанной и протестированной функциональности, образует растущую подсистему, которая также должна быть протестирована.

На каждой последующей итерации важность регрессионного тестирования повышается. Верификация и валидация могут проводиться для каждого приращения.

Инкрементная разработка предполагает определение требований, проектирование, создание и тестирование системы по частям, что означает инкрементный рост функциональных возможностей продукта. Размер этих приращений варьируется: в одних методах они больше, в других меньше. Приращением функциональных возможностей может быть всего одно изменение экрана пользовательского интерфейса или новая опция запроса.

3 Виды тестирования



СТАТИЧЕСКОЕ И ДИНАМИЧЕСКОЕ ТЕСТИРОВАНИЕ

Существует два класса высокоуровневого тестирования: статическое и динамическое. Тестирование, которое проводится после выполнения кода тестируемой системы, называется **динамическим**. Тестирование, которое может быть проведено без выполнения кода тестируемой системы, называется **статическим**.

Статическое тестирование может проводиться даже до того, как код программного обеспечения будет написан. Статическое тестирование позволяет находить и предотвращать дефекты путем анализа рабочих продуктов, таких как требования, проектная документация и исходный код. Это упрощает и удешевляет процесс внесения исправлений. Вместо того чтобы обнаруживать те же дефекты на более поздних этапах при выполнении тестов, они исправляются на уровне документации. В целях статического тестирования проводится **ревью**, которое предполагает ручную проверку рабочих продуктов проекта.

В проектах могут применяться следующие типы ревью:

- ***Неформальное ревью***
  Например, проверка работы коллег, совместная проверка, парное ревью.

Процесс неформального ревью **не формализован** и может быть довольно прост, как, например, в случае проверки работы коллег, когда тестировщик проверяет тест-кейсы своего товарища или когда руководитель группы проверяет результаты работы одного из своих сотрудников. В этом случае документирование результатов рецензирования не является обязательным, но иногда проводится.

Основная цель неформального ревью — выявление дефектов. Обычно оно не требует больших затрат, поскольку формальный процесс отсутствует, но может быть достаточно эффективным в зависимости от навыков и мотивации рецензента.

- ***Пошаговый разбор***

  Пошаговый разбор обычно используется для проверки черновых вариантов рабочих продуктов. Например, в случае реализации нового дизайна пошаговый разбор позволяет команде лучше понять подход, используемый при разработке. В свою очередь, участники могут дать рекомендации по процессу разработки.

  Основными целями здесь могут быть поиск дефектов и усовершенствование программного продукта, а возможными дополнительными целями могут стать обучение участников и достижение консенсуса.

- ***Техническое ревью***

  Техническое ревью проводится в рамках четко определенного процесса, в ходе которого фиксируются найденные ошибки и намечаются действия, которые следует предпринять. Ревью организуется в формате официального совещания с назначением координатора (который не должен быть автором проекта). В идеале этот человек является экспертом по проведению ревью.

  Ревьюеры должны обладать техническими знаниями в своих областях, имеющих отношение к рассматриваемому рабочему продукту.

  Подготовка к совещанию при формальном ревью является обязательной, при этом обычно заполняются чек-листы и составляется отчет, хотя эти шаги уже опциональны.

  Техническое ревью обычно проводится в следующих целях: обсуждение, принятие решений, оценка альтернатив, поиск дефектов, решение технических проблем или проверка соответствия спецификациям и стандартам.

- ***Инспекция***

  Инспекция представляет собой самый формализованный вид ревью и проводится для конкретного рабочего продукта. Впервые инспекция была

проведена в компании IBM в начале 1970-х годов Майклом Фэганом (Michael Fagan) и получила высокую оценку как одно из наиболее значительных усовершенствований в процессе разработки.

Для проведения инспекции требуется опытный модератор, который не должен быть автором проекта, и все его функции должны быть определены до начала инспекции.

Инспекция проводится в соответствии с формальным процессом, описанным в правилах и чек-листах; этот процесс также должен включать критерии входа и критерии выхода. Осуществляется сбор метрик, которые используются для оптимизации процессов и доработки документов.

Подготовка перед совещанием очень важна, а отчет об инспекции с перечислением результатов является обязательным компонентом, так же как и формальный процесс последующих действий.

Основной целью инспекции является выявление дефектов, однако инспекции также могут использоваться для инициирования мероприятий по оптимизации на основе полученных метрик.

Динамическое тестирование может проводиться только после выполнения кода. Оно проверяет корректность функционирования продукта и его соответствие спецификациям. Оно также помогает выявить любые ошибки выполнения программы, которые невозможно обнаружить во время статического тестирования. Такое тестирование можно начинать сразу после создания минимального фрагмента кода, который может выполняться независимо. На разных уровнях завершенности кода объем тестирования будет меняться. Динамическое тестирование можно разделить на несколько уровней. Каждый уровень представляет собой группу тестовых мероприятий, которые организованы и управляются совместно. Выделяют такие уровни динамического тестирования:

- компонентное/модульное тестирование;
- тестирование интеграции компонентов;
- системное тестирование;
- системное интеграционное тестирование;
- приемочное тестирование.

Каждый уровень динамического тестирования требует соответствующей тестовой среды. Например, при тестировании компонентов разработчики обычно используют локальную среду разработки, а для приемочного идеальным вариантом является тестовая среда, симулирующая продакшен.

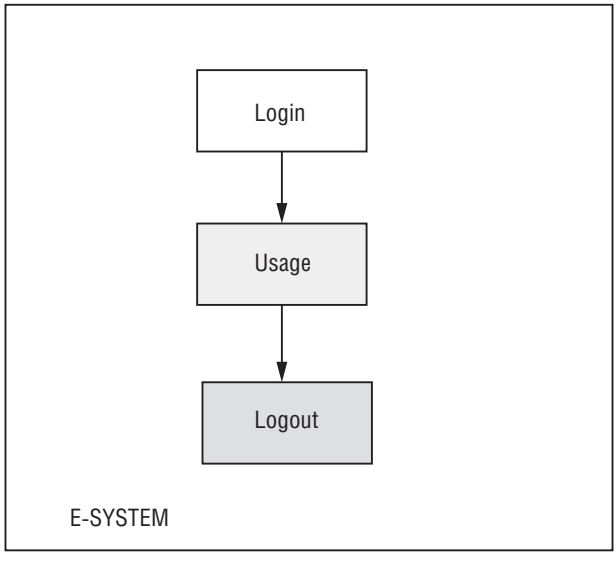
Чтобы лучше разобраться в уровнях тестирования, рассмотрим пример приложения под названием E-SYSTEM. Оно позволяет клиентам проверять потребление электроэнергии онлайн.

В этом приложении всего три компонента/модуля.

Вход в систему (Login) — используется для входа клиентов в систему с помощью логина/пароля.

Панель потребления (Usage) — предназначена для отображения текущих показаний счетчика электроэнергии клиента.

Выход (Logout) — используется для выхода клиента из приложения.



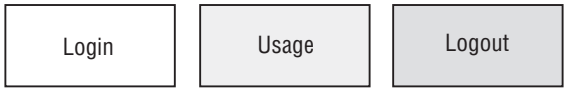
КОМПОНЕНТНОЕ, ИЛИ МОДУЛЬНОЕ, ТЕСТИРОВАНИЕ

Компонентное тестирование (также известное как юнит-тестирование, или модульное тестирование) проводится для компонентов, которые можно тестировать по отдельности.

Такое тестирование часто выполняется независимо от остальных компонентов системы. Оно может покрывать как функциональные (например, правильность расчетов) и нефункциональные (например, поиск утечек памяти) характеристики, так и свойства структуры (например, проверку инструкций и решений в коде).

Если обратиться к нашему примеру приложения E-SYSTEM, то разработчики будут создавать компоненты Login, Usage и Logout по отдельности. Тестирование каждого модуля также будет проводиться независимо от других.

Модульное тестирование обычно выполняется разработчиком сразу после написания кода компонента.



ПРИМЕЧАНИЕ

- Цель **компонентного тестирования** — убедиться, что код компонента соответствует спецификации, прежде чем проводить его интеграцию с другими компонентами.
- Оно выполняется разработчиками, писавшими код компонента.
- Оно проводится в среде разработки.
- Найденные дефекты устраняются по мере их обнаружения без прохождения процесса управления дефектами.

ТЕСТИРОВАНИЕ ИНТЕГРАЦИИ КОМПОНЕНТОВ

После разработки отдельных компонентов необходимо провести их интеграцию и тестирование. Тестирование интеграции компонентов выполняется после того, как отдельные компоненты объединены в группу и протестированы по отдельности. Цель этого уровня тестирования — проверить совместную работу программных компонентов и выявить ошибки взаимодействия между интегрированными модулями.

Тестирование интеграции компонентов часто входит в обязанности разработчиков и проводится в среде разработки.

Существуют три основные стратегии интеграции.

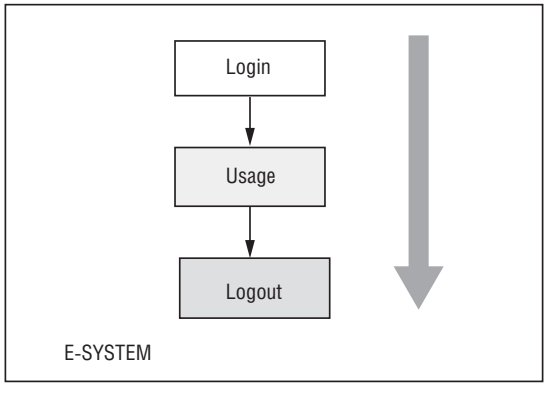
Интеграция по принципу «большого взрыва»

В этом случае интеграция всех компонентов проводится на одном этапе, в результате чего получается законченная система.

Если для нашего примера использовать интеграцию по принципу «большого взрыва», то все модули — Login, Usage и Logout — должны быть готовы, а затем сразу же соединены.

Интеграция сверху вниз

В этом случае система строится поэтапно, начиная с компонентов, которые вызывают другие компоненты. Компоненты, вызывающие другие компоненты, обычно в иерархии располагаются выше тех, к которым обращаются. Интеграционное тестирование по принципу «сверху вниз» позволит тестировщику оценить интерфейсы компонентов, начиная с тех, которые находятся «сверху».



ИНТЕГРАЦИЯ СВЕРХУ ВНИЗ

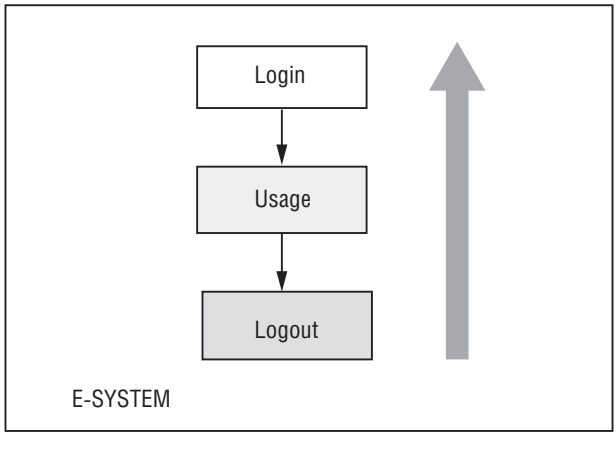
Структуру управления программой можно представить, как показано на схеме выше. Компонент Login вызывает Usage. Компонент Usage вызовет компонент Logout. Порядок интеграции будет следующий:

- Login, Usage;
- Usage, Logout.

В этом случае при тестировании первого сценария, если компонент Usage не готов, будет создана скелетная реализация компонента, называемая заглушкой (stub). **Заглушка — это пассивный компонент, вызываемый другими компонентами**. В данном случае заглушка будет возвращать только два значения: «успешный вход» для верной комбинации «логин/пароль» и «ошибка входа» — для неверной. Этих двух значений достаточно для тестирования модуля Login.

Интеграция снизу вверх

Это полная противоположность интеграции «сверху вниз», и компоненты интегрируются в порядке снизу вверх.



ИНТЕГРАЦИЯ СНИЗУ ВВЕРХ

Порядок интеграции будет следующим:

- Logout, Usage;
- Usage, Login.

При тестировании по первому сценарию в рамках интеграции снизу вверх, если компонент Usage еще не создан, вместо него следует использовать компонент,

называемый **драйвером**. **Драйверы — это активные компоненты, которые вызывают другие компоненты.** Как правило, они более сложны, чем заглушки.

Чтобы упростить изоляцию дефектов и их обнаружение на ранней стадии, обычно используют инкрементную интеграцию, а не интеграцию по принципу «большого взрыва». Анализ рисков наиболее сложных интерфейсов может помочь сосредоточиться на интеграционном тестировании.

ПРИМЕЧАНИЕ

Заглушка (stub) — это скелетная или созданная под конкретную цель реализация программного компонента, используемая для разработки или тестирования другого компонента, вызывающего его или зависящего от него иным образом. Она заменяет вызываемый компонент.

Драйвер — это программный компонент или инструмент тестирования, заменяющий компонент, который осуществляет управление и/или вызов тестируемого компонента или системы.

ПРИМЕЧАНИЕ

- Целью **тестирования интеграции компонентов** является выявление дефектов во взаимодействии между интегрированными компонентами. Оно проводится после модульного тестирования.
- Оно выполняется разработчиками, написавшими компоненты.
- Оно проводится в среде разработки.

СИСТЕМНОЕ ТЕСТИРОВАНИЕ (ST)

Системное тестирование направлено **на изучение поведения и возможностей системы или продукта в целом** и часто применяется к сквозным задачам, которые может выполнять система, и нефункциональному поведению, которое она демонстрирует при выполнении этих задач.

Тесты для такого тестирования создаются на основе бизнес-требований, которые обычно разрабатываются бизнес-аналитиками в рамках проектов.

Системное тестирование часто позволяет получать информацию, которая используется стейкхолдерами для принятия решения о релизе. Кроме того,

системное тестирование может проводиться в соответствии с законодательными или нормативными требованиями или стандартами.

Системное тестирование должно главным образом касаться общего, сквозного поведения всей системы. В ходе него должны проверяться **как функциональные, так и нефункциональные** требования к системе.

Команда тестировщиков обычно проводит системное тестирование в тестовой среде.

ПРИМЕЧАНИЕ

- Целью системного тестирования является валидация поведения системы в целом.
- Оно выполняется группой тестирования.

СИСТЕМНОЕ ИНТЕГРАЦИОННОЕ ТЕСТИРОВАНИЕ (SIT)

Системное интеграционное тестирование оценивает взаимодействие между различными системами и выполняется в основном после системного тестирования (ST). Оно может также охватывать взаимодействие с внешними интерфейсами (например, веб-сервисами). SIT-тестирование выявляет любые недочеты во взаимодействии системы с другими приложениями.

В то время как тест-кейсы ST ориентированы на отдельную тестируемую систему, тест-кейсы SIT проверяют передачу данных в другие системы или получение данных из них.

Системное интеграционное тестирование выполняется в тестовой среде командой тестировщиков. Тестировщики, выполняющие системное интеграционное тестирование, должны понимать архитектуру системы и иметь возможность влиять на процесс планирования интеграции.

Если в нашем примере приложение E-SYSTEM передает данные из Usage (Панели потребления) во внешнее приложение Billing (Выставление счетов), которое обрабатывает эту информацию для формирования счета-фактуры для клиента, то проверка правильности передачи данных в приложение Billing для обработки и будет являться предметом системного интеграционного тестирования.