Заглушки: виды, способы использования. Плюсы и минусы тестирования на заглушках

Если говорить о заглушке БД, то это эмуляция успешных запросов в базу данных. Это нужно для тестирования кода без использования базы данных. Если вы тестируете, например, формирование строки для вставки в БД, то сама БД вам не нужна.

Заглушка — это компонент, который не делает ничего кроме того, что объявляет себя и набор принимаемых параметров. Заглушка содержит минимум кода, необходимого для ее компиляции и компоновки с остальными компонентами.

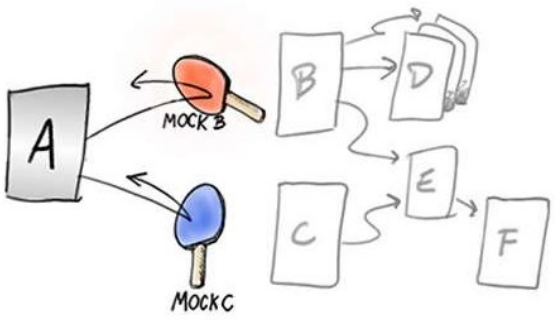
Заглушка - некая сущность, которая заменяет реальную, работа которой в данном случае не важна. В идеале заглушками могут быть вообще все сущности, которые не имеют никакого отношения к отлаживаемому в данный момент коду, и заменяться по мере необходимости по схеме "отладил кусок - заменил одну заглушку на код, отладил бывшую заглушку - приступаем к следующей", и т.д.

Чтобы в тестировании объектов, которые используют данную зависимость(интерфейс) не работать с реальной базой данных, можно создать *объект-заглушку*. Данная заглушка будет возвращать заранее описанные данные, и для пользователей-объектов интерфейса разницы никакой не будет, так как они получат ожидаемые данные, но при этом не будет зависимости от реальной базы данных.

Виды заглушек:

Существует несколько видов объектов, которые позволяют симулировать поведение реальных объектов во время тестирования:

1. **Dummy** — пустые объекты, которые передаются в вызываемые методы, но не используются. Предназначены лишь для заполнения параметров методов.
2. **Fake** — объекты, которые имеют реализации, но в таком виде, который делает их неподходящими для использования в рабочей ситуации.
3. **Stub** — предоставляют заранее заготовленные ответы на вызовы во время теста и не отвечают ни на какие другие вызовы, которые не требуются в тесте.
4. **Mock** — объекты, которые заменяют реальный объект в условиях теста и позволяют проверять вызовы своих методов. Содержат заранее подготовленные описания вызовов, которые они ожидают получить. Применяются в основном для тестирования поведения пользователя.



Относительно веб-сервисов можно сказать, что мок-сервис позволяет переопределить реальный сервис и подставить вместо него упрощенную реализацию, которая работает нужным разработчику образом и дает доступ к собственным данным и

настройкам. Мы как бы создаем у себя аналог удаленного веб-сервиса, который отвечает на вызовы нашего приложения.

Правда, тут возникают две сложности.

Во-первых, **моки надо создать**. Надо устанавливать дополнительный софт, описывать логику, загружать нужные данные в макет ответа. Не то чтобы это сложно, но это требует усилий и времени. Может быть, кому-то будет проще сохранить результат вызова в файл и при тестировании обращаться не к заглушке, а к файлу. В небольших проектах с малым количеством тестов такой подход вполне оправдан.

Во-вторых, **веб-сервис может поменяться**. Например, программист написал рабочую логику для заглушки, а через какое-то время веб-сервис стал работать по-другому (а разработчика, как часто бывает, об этом никто не уведомил). О том, что в работе веб-сервиса что-то поменялось он узнал, когда выкатил обновление на рабочую базу.

Решение простое — для сервиса, который может быть подвержен изменениям нужно сделать набор дымовых тестов (smoke-тесты). Дымовые тесты проверяют, что основные методы сервиса не изменились и обращение к ним не вызывает ошибок.

Использование заглушек

Разработка для внедрения зависимости

Для использования заглушек приложение должно быть разработано таким образом, чтобы различные компоненты не зависели друг от друга, а только от определений интерфейса. Вместо соединения во время компиляции компоненты соединяются во время выполнения. Этот шаблон позволяет создать надежное и легко обновляемое программное обеспечение, поскольку изменения, как правило, не выходят за границы компонентов. Им рекомендуется пользоваться, даже если вы не используете заглушки. При написании нового кода легко использовать шаблон внедрения зависимости. При написании тестов для существующего программного обеспечения может потребоваться выполнить его рефакторинг. Если это непрактично, можно рассмотреть возможность использования оболочек.

Использование заглушек для изоляции частей приложений друг от друга при модульном тестировании

Типы заглушек — одна из двух технологий, которые платформа Microsoft Fakes предоставляет для более простого изолирования тестируемого компонента от других вызываемых компонентов. Заглушка — это небольшая часть кода, которая заменяет собой другой компонент во время тестирования. Преимущество использования заглушки заключается в том, что она возвращает последовательные результаты, упрощая написание теста. Тесты можно выполнять, даже если другие компоненты пока не работают.

Для использования заглушек необходимо написать компонент таким образом, чтобы он использовал только интерфейсы, а не классы для ссылки на другие части приложения. Это рекомендуемая практика разработки, поскольку при внесении изменений в одну часть, скорее всего, не потребуется вносить изменения в другую. При тестировании это позволяет заменить заглушку на реальный компонент.

Поскольку предполагается, что заглушки могут структурировать код, они обычно используются для изолирования одной части приложения от другой. Для изолирования ее от других сборок, которыми вы не можете управлять, например System.dll, обычно используются оболочки.

Создание заглушек

Вы отделили класс, который необходимо протестировать, от других используемых им компонентов. Такое отделение позволяет не только сделать приложение более надежным и гибким, но и подключить тестируемый компонент к реализациям заглушки интерфейсов для тестирования.

Достаточно написать заглушки как классы обычным способом. Однако Microsoft Fakes предлагает более динамический способ создания наиболее подходящей заглушки для каждого теста.

Чтобы использовать заглушки, необходимо сначала создать типы заглушек из определений интерфейса.

Создание теста с заглушками
Заглушки также создаются для методов получения и задания свойств, для событий и для универсальных методов.

Проверка значений параметров

Можно проверить, что компонент передает правильные значения, когда вызывает другой компонент. Можно поместить утверждение в заглушку или сохранить значение и проверить его в основной части теста. Пример:

Заглушки для различных видов членов типа

Методы
Если не предоставить заглушку для функции, Fakes создает функцию, возвращающую значение по умолчанию возвращаемого типа. Для чисел значением по умолчанию является 0, а для типов классов — null (C#) или Nothing (Visual Basic).

Свойства

Если не предоставить методы-заглушки для метода получения или задания свойства, Fakes создает заглушку, хранящую значения, чтобы свойство заглушки выступало в роли простой переменной.

События
События представляются как поля делегата. В результате любые события, замененные заглушками, могут вызываться путем вызова резервного поля события. Рассмотрим следующий интерфейс, который требуется заменить заглушкой.

Универсальные методы

Чтобы заменить универсальные методы заглушками, предоставьте делегат для каждого требуемого экземпляра метода. Например, рассмотрим следующий интерфейс, содержащий универсальный метод.

Заглушки виртуальных классов
В предыдущих примерах заглушки создавались из интерфейсов. Заглушки также можно создавать из класса с виртуальными или абстрактными членами. Пример:
Заглушки отладки

Типы заглушек призваны обеспечить бесперебойную отладку. По умолчанию отладчик должен обходить любой создаваемый код, чтобы сразу переходить к пользовательским реализациям членов, которые присоединены к заглушке.

Ограничения заглушки

Сигнатуры методов с указателями не поддерживаются.

Запечатанные классы или статические методы не могут быть заменены заглушками, поскольку типы заглушек зависят от диспетчеризации виртуальных методов. В таких случаях используйте типы оболочек, которые описываются в статье Использование оболочек совместимости для изоляции приложения от других сборок при модульном тестировании.

Добавляйте файлы заглушек в репозиторий.

Не бойтесь удалять и перезаписывать файлы заглушек.

Управляйте файлами заглушек разумно. Вы можете переиспользовать один файл в нескольких тестах или обеспечить индивидуальный файл для каждого теста. Пользуйтесь этой возможностью для ваших потребностей. Но обычно использование одного файла с разными режимами доступа — плохая идея.

Особенности тестирования на заглушках

Пример:

Тестирование компонента заключается в подаче входных данных в его интерфейс, ожидании ответа компонента и проверки полученных результатов. В процессе работы компоненты часто пользуются другими компонентами, передают им данные и используют полученную от них информацию:

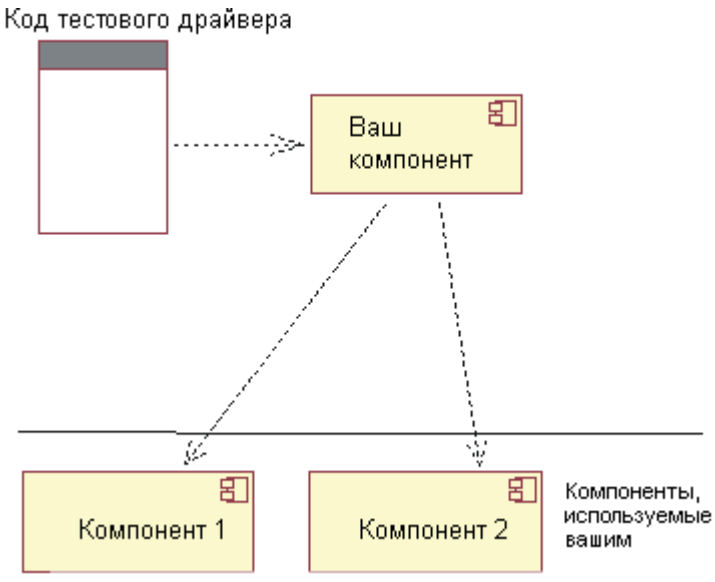


Рисунок 1: Тестирование реализованного компонента

Эти другие компоненты могут вызывать сложности при тестировании:

1. Они могут быть еще не реализованы.
2. В них могут быть ошибки, нарушающие работу тестов, причем на выяснение того, что ошибка была вызвана не тестируемым, а зависимым компонентом, может уйти много времени и усилий.
3. Компоненты могут затруднять тестирование тогда, когда это действительно надо. Например, если компонент - коммерческая база данных, у вас может оказаться недостаточно лицензий для всех пользователей. Другой пример: компонент может представлять собой аппаратное обеспечение, доступное только в определенные промежутки времени в определенной лаборатории.
4. Компоненты могут настолько замедлить тестирование, что станет невозможно выполнять тесты достаточно часто. Например, инициализация базы данных может занимать пять минут.
5. Может быть сложно поставить компонент в условия, при которых он выдаст ожидаемый от него результат. Например, перед вами может стоять задача протестировать обработку сообщения "диск переполнен" для всех методов, осуществляющих запись на диск. Как гарантировать, что диск будет заполнен именно на момент вызова нужного метода?

Плюсы использования заглушек

Во избежание возникновения таких сложностей, описанных выше можно пользоваться заглушками компонентов (другое название заглушек - **имитаторы**). Заглушки должны вести себя так же, как вели ли бы себя реальные компоненты, на тех входных данных, которые они могут получить в ходе тестирования. Однако это только задача-минимум: в качестве заглушек можно использовать и полноценные **эмуляторы**, с максимальной точностью имитирующие поведение реального компонента. Например, довольно часто имеет смысл создание программных заглушек для аппаратного обеспечения. Они будут вести себя точно так же, как аппаратное обеспечение, только работать медленнее. Применение таких заглушек упрощает отладку, снимает проблему недостатка единиц оборудования, и наконец, ими можно пользоваться еще до завершения разработки оборудования.

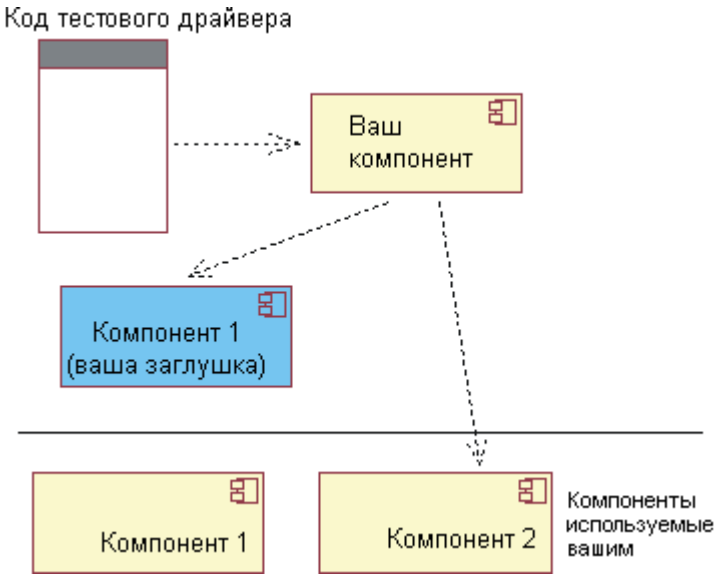


Рисунок 2: Тестирование компонента с заглушкой вместо компонента, от которого он зависит

Причины использования заглушек веб-сервисов:

1. Тесты замедляются. Если поставщик сервиса находится далеко, сетевая среда нестабильная и при вызове сервиса происходит задержка, то время прохождения тестов может значительно увеличиваться.
2. Тесты работают нестабильно. Веб-сервисы могут быть не всегда доступны из-за технических обновлений, подвержены перегрузкам или ошибкам сетевых протоколов.
3. Тесты покрывают не все возможные варианты ответов сервиса. Не всегда есть возможность получить некоторые ответы от реального веб-сервиса и промоделировать все рабочие ситуации — следовательно, максимально полно протестировать взаимодействие.
4. Доступ к рабочим веб-сервисам ограничен. Часто встречается ситуация, когда рабочие сервисы недоступны из тестового контура, в котором программисты ведут разработку. Это делает как по причине безопасности, так и для того, чтобы не подвергать лишней нагрузке рабочее окружение.
5. Из-за ошибок разработчика могут быть подвергнуты риску реальные данные. Например, есть веб-сервисы, которые позволяют добавлять или изменять данные в удаленной системе. Ошибка при вызове такого сервиса может привести к потере данных.

У заглушек есть два недостатка:

6. Заглушки могут быть дорогостоящими (это особенно актуально для эмуляторов). Поскольку заглушки сами по себе представляют программное обеспечение, они нуждаются в обслуживании.
7. Заглушки могут маскировать ошибки. Например, предположим, что ваш компонент пользуется тригонометрическими функциями, но к настоящему моменту соответствующая библиотека еще не разработана. Вы пользуетесь тремя тестами для вычисления синуса трех углов: 10 градусов, 45 градусов и 90 градусов. Вы находите нужные значения с помощью калькулятора и строите заглушку функции синуса, которая возвращает на эти входные параметры значения 0,173648178, 0,707106781 и 1,0, соответственно. Все прекрасно работает до момента интеграции компонента в готовую библиотеку, в которой функция синуса принимает на входе значения в радианах и возвращает значения -0,544021111, 0,850903525 и 0,893996664. В итоге вы обнаруживаете ошибку в коде позже и потратив больше усилий, чем хотелось бы.

Заглушки и практика разработки программного обеспечения

Иногда заглушки создаются только потому, что реальный компонент еще не доступен на момент тестирования. Во всех остальных случаях заглушки следует сохранять после завершения разработки. Скорее всего, тесты, поддерживаемые заглушками, будут важны при обслуживании продукта. Поэтому качество заглушек должно быть выше, чем качество кода на выброс. Хотя к качеству заглушек не предъявляются настолько жесткие требования, как к коду продукта (например, большинство заглушек не нуждается в собственных тестах), в дальнейшем некоторые из них придется обслуживать в процессе изменения продукта. Если обслуживание будет требовать слишком больших усилий, заглушки будут выброшены, и инвестиции в них будут потеряны.