Определением Unit тестирования является кусок кода (обычно методом), который вызывает другой кусок кода и проверяет правильность некоторых предположений позже. Если предположения оказываются неверными, Unit тест был провален. «Unit» это метод или функция.

Юнит-тест представляет собой автоматизированный кусок кода, который вызывает тестируемый метод или класс, а затем проверяет некоторые предположения о логического поведения этого метода или класса. Это полностью автоматизировано, надежно, легко читаемо и сопровождаемо.

Unit тест должен быть:

1. Автоматизированным и повторяемым.

2. Простым в реализации

3. После написания он должен остаться для последующего использования.

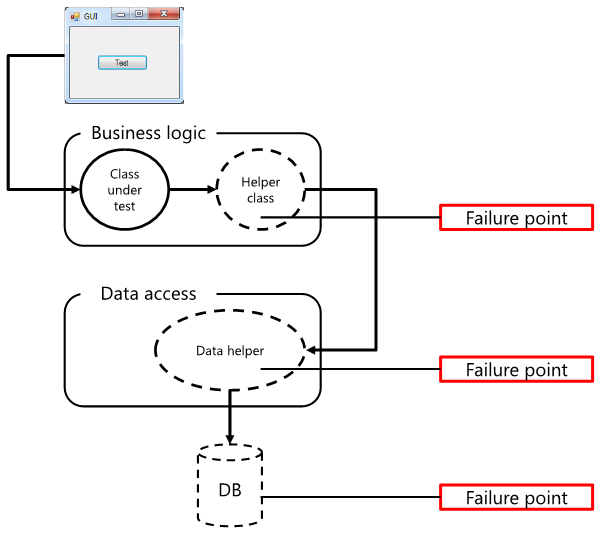
4. Кто угодно в команде должен иметь возможность запустить Unit тест.

5. Должен запускаться одним нажатием кнопки.

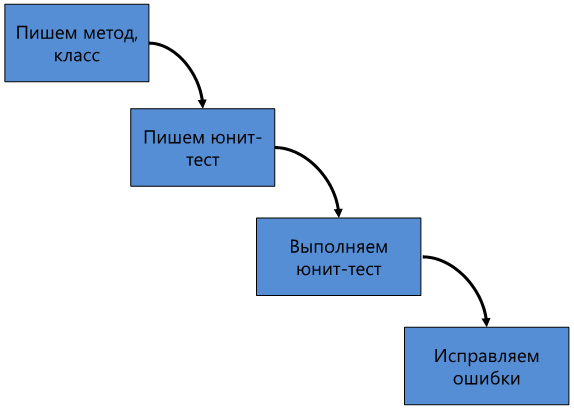
6. Должен выполнятся быстро.

Интеграционное тестирование означает тестирование двух или более зависимых модулей программного обеспечения в группе.

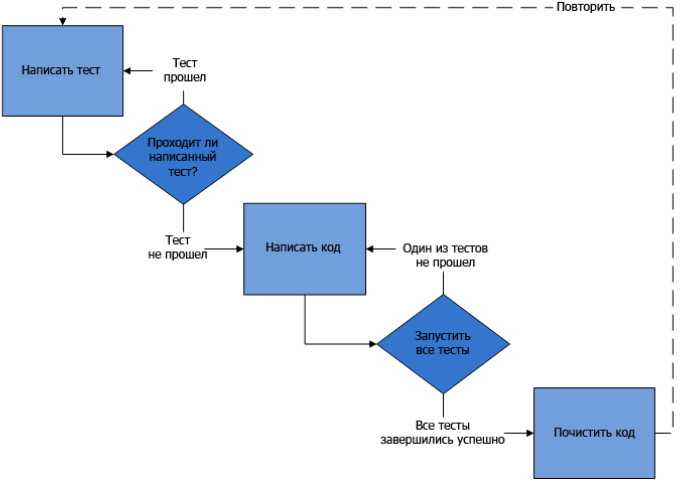
Вы можете иметь много критических точек в интеграционном тесте. Все модули работают вместе, и каждый из них может быть неисправен, делая поиск источника ошибки сложнее.



Как только мы знаем как писать структурированные, сопровождаемые и хорошие тесты с использованием Unit framework, следующий вопрос, когда писать тесты. Многие считают, что лучшее время, чтобы писать тесты для программного обеспечения после того, как программа была написана, но большинство людей предпочитают писать юнит-тесты перед производством кода. Этот подход называется "тест вначале" или разработка через тестирование (TDD - test-driven development).



**Традиционный способ написания юнит-тестов**



**TDD с высоты птичьего полета**

Техника TDD достаточно проста:

1. Написать тест, который доказывает неработоспособность конечного продукта. Мы должны писать тест, как будто у нас уже есть рабочий код, так что ошибка теста означает недоработку производственного кода.

2. Пройти тест, с помощью написания нового кода, который будет отвечать всем ожиданиям нашего теста. Он должен быть написан так просто, как это возможно.

3. Рефакторинг кода. Если тест пройден, вы можете переходить к следующему юнит-тесту или почистись код, чтобы сделать его более удобным для чтения, убрать дублирование кода и так далее. Рефакторинг может быть сделан после написания нескольких тестов или после написания каждого теста. Это важная практика, поскольку она обеспечивает чистоту вашего кода, читабельность и сопровождение, в то же время прохождение всех ранее написанных тестов.

Рефакторинг - изменение кода без изменения его функциональности.

1. Если вы когда-либо переименовывали метод, вы сделали рефакторинг.

2. Если вы когда-либо разделяли большой метод на несколько меньших методов, вы сделали рефакторинг своего кода.

3. Код по-прежнему делает то же самое, но становится легче для сопровождения, более читаемый и изменяемый.

Правильно реализованное TDD может повысить качество вашего кода, уменьшить количество ошибок, повысить уверенность в коде, сократить время для поиска ошибок, улучшить дизайн кода, и сделать вашего менеджера счастливей.

Юнит-тест фреймворки состоят из библиотек и модулей, которые помогают разработчикам тестировать свой код с помощью юнит-тестов.

Есть более чем 150 юнит-тест фреймворков, практически по одному на каждый язык программирования. .Net имеет 9 различных юнит-тест фреймворков. Среди них, NUnit является стандартом.

В совокупности, эти юнит-тест фреймворки называют xUnit фреймворками, потому что их имена обычно начинаются с первых букв языка, для которых они были построены. Возможно вы будете встречать CppUnit для C++, JUnit для Java, NUnit для .Net и HUnit для языка программирования Haskell. Не все из них следуют этим правилам именования, но большинство их используют.