Модульное тестирование позволяет проверить на корректность отдельные модули исходного кода программы (рис. 1). Этот тип тестирования обычно выполняется программистами в процессе кодирования. Существует даже метод разработки посредством тестирования, когда перед разработкой какой-либо полезной функциональности создается её модульный тест. Идея состоит в том, чтобы писать тесты для каждой сложной функции. В случае если необходимо разработать модульный тест для класса, обычно для каждого его метода пишется тест. Такой тест проверяет все возможные способы поведения класса (в соответствии со спецификацией) и сообщает об обнаруженных ошибках. Это позволяет быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже написанных и протестированных местах программы, а также облегчает локализацию и устранение таких ошибок. Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части — работоспособны.

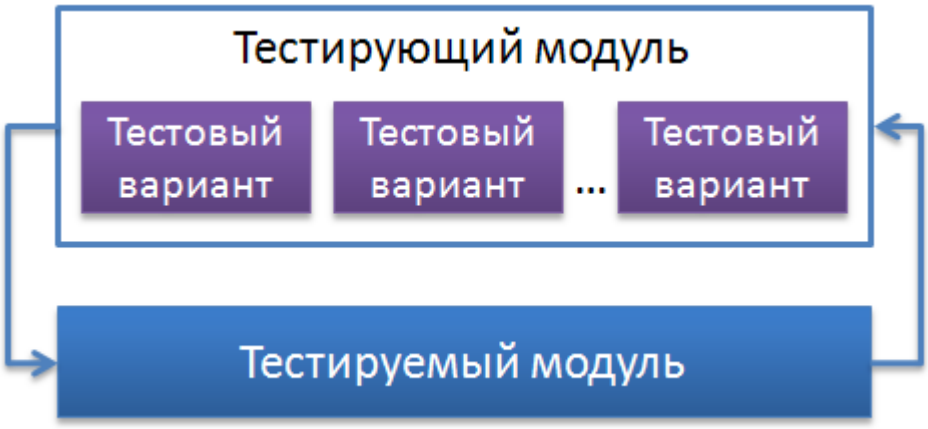

Рис. 1. Пример организации модульного тестирования

При внедрении технологии модульного тестирования можно получить следующие преимущества:

- ускорение и удешевление разработки;

> Модульное тестирование позволяет упростить процесс отладки кода. Часто ошибки в низкоуровневом коде приводят к исправлениям в высокоуровневом коде. В итоге в процессе исправления не причин, а следствий затрачиваются значительные усилия, усложняется код. Модульное тестирование способствует раннему обнаружению некорректного поведения в низкоуровневых элементах архитектуры ПО.

- поощрение изменений;

Модульное тестирование позволяет проводить рефакторинг, будучи уверенным, что код, покрытый тестами, по-прежнему работает корректно. Это поощряет разработчиков к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений.

- упрощение интеграции;

Модульное тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируются отдельные части программы, затем программа в целом.

- документирование кода;

Модульные тесты можно рассматривать как «живой документ» для тестируемого кода. Разработчики, которые не знают, как использовать данный код, могут использовать соответствующий тест в качестве примера.

- отделение интерфейса от реализации;

Поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на связанные с ним. Необходимость разработки модульного теста стимулирует уменьшать число таких зависимостей, чтобы логика теста была максимально простой и не требовала существенных усилий по разработке. Это приводит к менее связанному коду, минимизируя зависимости в системе. Что в свою очередь упрощает дальнейшую поддержку кода, т.к. минимизируется «эффект ряби» при внесении изменений.

В случае использования метода разработки посредством тестирования, также можно получить сокращение времени отладки и сократить длинные циклы тестирования на завершающем этапе проекта.

Модульное тестирование, как правило, может осуществляться на том же языке программирования, на котором ведётся разработка приложения. В настоящее время доступно много библиотек, содержащих реализацию программного окружения для организации модульного тестирования на разных языках программирования. Например, для тестирования программ, разработанных на языке C++, может использоваться библиотека CPPUNIT. Для модульного тестирования программ, разработанных на языке Java, обычно используют JUnit. Для семейства языков, поддерживающих разработку для платформы Microsoft .Net могут использоваться MSUnit и NUnit. Существуют аналогичные библиотеки и для других языков (Delphi – Dunit, Python – PyUnit и т.д.).

На уровне модульного тестирования проще всего обнаружить дефекты, связанные с алгоритмическими ошибками и ошибками кодирования алгоритмов, типа работы с условиями и счетчиками циклов, а также с использованием локальных переменных и ресурсов.

Ошибки, связанные с неверной трактовкой данных, некорректной реализацией интерфейсов, совместимостью, производительностью и т.п. обычно пропускаются на уровне модульного тестирования и выявляются на более поздних стадиях тестирования.