Рекомендации к главе отчета – модульное тестирование выбранной части кода (этап 3 курсовой работы)

1. Согласно заданию курсовой работы, для изучения модульного тестирования и инструментария, необходимо заранее определить часть кода приложения, примерно 300-400 строк исходного кода (желательно, чтобы проверке подвергались несколько модулей). Отобранный код приложения в отчет можно не включать, но надо указать, какие именно модули выбраны для тестирования. Сделать это надо таким образом, чтобы человек, получивший доступ к коду, мог легко найти эти модули.

2. Напишите название, версию и общие пояснения о выбранном вами фреймворке тестирования или отдельном инструменте, наборе скриптов, библиотек и т.п., которые планируете использовать для модульного тестирования.

3. Опишите процедуру установки инструмента в вашей среде или укажите, каким образом можно найти данный инструмент в интегрированной среде и инициировать его запуск для проверки конкретного модуля/модулей с набором тестов.

4. Опишите стратегию тестирования отобранных модулей. В стратегии надо пояснить общий подход к проверке отобранных модулей, указать проверки, которые планируются реализовать в модульных тестах.

Например, если в модуле вы проверяете метод для открытия файла, то в стратегии необходимо указать, что вы будете проверять открытие тестового файла, параметры которого должны совпадать с ожидаемыми (формат, минимальный и максимальный размер и др.), файл с неправильными атрибутами доступа, реакцию программы на отсутствие файла при обращении к нему и другие проверки, в зависимости от реализации конкретного метода.

Тесты должны учитывать не только, так называемые, положительные тесты, когда тестовые данные не выходят за границы допустимых значений, или последовательность действий не нарушается при обработке данных. Необходимо написать и «отрицательные тесты», проверяя реакцию программы на данные, лежащие за пределами границ допустимых значений и т.п.

5. Если вы планируете разработать автоматизированные тесты, то дополнительно опишите процедуру их выполнения, получения и сохранения результатов и др. аспекты авто-тестирования.

6. Напишите модульные тесты, текст которых поместите в приложение к отчёту.

7. Опишите процедуру запуска набора тестов, получения результатов тестирования, укажите в каком виде сохраняются результаты тестирования.

8. Выполните ваш программный проект вместе с модульными тестами, приведите результаты исполнения в виде протокола / скриншотов / логов / отчета о прогоне и др.

9. Опишите анализ проваленных тестов и причины, почему это произошло. Опишите исправления, сделанные в коде программы (или в коде тестов!), повторите проверку программы модульными тестами, опишите окончательный результат проверки кода модульными тестами.

10. Обязательно отметьте, измеряли ли вы покрытие кода тестами; если измеряли, то проанализируйте результат измерения.

11. Добавьте таблицу с количеством ошибок, найденных с помощью тестов и остаточное количество ошибок (если такие будут! Предполагается, что для выбранных модулей кода все найденные ошибки должны быть исправлены).

12. В конце раздела должны быть краткие выводы и ваши предложения относительно места модульного тестирования в жизненном цикле программного проекта. Опишите также ошибки какого типа могут быть найдены при применении такого метода проверки кода.

Основные принципы создания «хорошего теста»

1) Тесты должны быть независимыми и повторяемыми.
2) Тесты должны быть хорошо организованы и отражать структуру тестируемого кода.
3) Тесты должны быть переносимыми и переиспользуемыми (portable and reusable).
4) Когда тесты терпят неудачу, они должны предоставить как можно больше информации о проблеме. При некритических сбоях хорошо бы иметь возможность продолжить текущий тест с выдачей сообщения об ошибке в конкретном месте теста (зависит от фреймворка).
5) Удобство задания тестов для запуска без лишнего перечисления (зависит от фреймворка).
6) Тесты должны быть быстрыми (без лишних зависимостей между ними при создании/разрушении тестовой среды).
===== *** =====

Те, кто не сталкивался еще с тестовыми фреймворками и принципами организации модульного тестирования, рекомендую прочитать в Интернете про Google Test.

Знакомство с тестовым фреймворком (Google Test) и библиотекой специальных мок-объектов (Google Mock) для организации модульного тестирования приложения

На странице **GoogleTest User's Guide** указаны ссылки и подсказка, что находится на той или иной странице документации:
https://github.com/google/googletest/blob/master/docs/index.md

Рекомендации о порядке чтения.

Ключевыми документами для изучения являются (возможный порядок прочтения):
ссылка на GoogleTestPrimer - краткое описание данного тестового фреймворка и описание подхода и требований к тестам,
ссылка на GoogleTestFAQ (часто задаваемые вопросы по Googletest),

ссылка на GoogleTestSamples (внимательно изучите исходный код хотя бы для первых 3-х примеров: список примеров с описаниями и тестируемый код + тесты),
ссылка на GoogleTestAdvanced - здесь описано про не часто используемые assertations, про то, как создавать сложные сообщения об ошибках, распространять (propagate) фатальные отказы, повторно использовать и ускорять ваши тестовые фикстуры, а также использовать различные флаги в ваших тестах.

ссылка на Mocking for Dummies - краткое описание как создавать mock объекты и использовать их в тестах; рекомендуется снова прочитать этот документ после попытки написать тесты и фиктивные объекты (мок-объектов) и их выполнения, чтобы понять принципы gtest и gmock,

которые начали применять на практике;

ссылка на Mocking Cookbook – включает советы и подходы к типичным случаям использования mock подхода;

ссылка на Mocking Cheat Sheet - краткое изложение правил gmock из всех документов, но в нем отсутствуют некоторые вещи, которые явно прописаны только в «руководстве для чайников»;

ссылка на Mocking FAQ – ответы на некоторые вопросы, специфичные для mock подхода.

Мок-объект реализует тот же интерфейс, что и реальный объект (так что его можно использовать в качестве него), но позволяет указать во время выполнения КАК объект будет использоваться и ЧТО он должен делать (какие методы будут вызываться? В каком порядке? Сколько раз? с какими аргументами? что они возвращают? и т. д.)

gMock - это библиотека (иногда её называют тоже «фреймворком») для создания и использования мок-классов.

Используя gMock:

- вы используете несколько простых макросов для описания интерфейса, который вы хотите имитировать, и они будут расширены до реализации вашего мок-класса;
- затем вы создаете несколько мок-объектов и определяете их ожидания и поведение, используя интуитивно понятный синтаксис;
- затем вы выполняете код, который использует мок-объекты. gMock уловит любое нарушение ожиданий, как только оно возникнет.

googletest - это среда тестирования, разработанная командой Testing Technology с учетом особых требований и ограничений Google. Может работать на Linux, Windows или Mac. Поддерживает любые тесты, а не только модульные тесты.

При использовании googletest вы начинаете с написания утверждений (assertions), которые представляют собой утверждения, проверяющие истинно ли условие. Результатом утверждения может быть успех, некритичный сбой или фатальный сбой (success, nonfatal failure, or fatal failure). Если происходит фатальный сбой, текущая функция прерывается; в противном случае программа продолжится в обычном режиме.

Тесты используют утверждения для проверки поведения тестируемого кода. Если тест завершается с ошибкой или имеет неудавшееся утверждение, то он не выполнен; в противном случае он выполнен успешно.

Набор тестов содержит один или несколько тестов. Вы должны сгруппировать свои тесты в наборы тестов, которые отражают структуру тестируемого кода. Если нескольким тестам в наборе тестов необходимо использовать общие объекты и подпрограммы, вы можете поместить их в test fixture class (тестовая фикстура).

Программа тестирования может содержать несколько наборов тестов.

Утверждения googletest - это макросы, похожие на вызовы функций. Вы тестируете класс или функцию, делая утверждения о ее поведении. Когда утверждение не выполняется, googletest распечатывает исходный файл утверждения и расположение номера строки вместе с сообщением об ошибке. Вы также можете указать собственное сообщение об ошибке, которое будет добавлено к сообщению googletest.

Основные принципы создания «хорошего теста» и что googletest делает для этого?

1. Тесты должны быть независимыми и повторяемыми.
googletest изолирует тесты, запуская каждый из них на другом объекте. Когда тест не проходит, googletest позволяет вам запускать его изолированно для быстрой отладки.

2. Тесты должны быть хорошо организованы и отражать структуру тестируемого кода.
googletest группирует связанные тесты в наборы тестов, которые могут обмениваться данными и подпрограммами. Этот общий шаблон легко распознать, и тесты легко поддерживать.

3. Тесты должны быть переносимыми и переиспользуемыми (portable and reusable). У Google много кода, нейтрального к платформе; его тесты также не должны зависеть от платформы. googletest работает на разных ОС, с разными компиляторами, с исключениями или без них, поэтому тесты googletest могут работать с различными конфигурациями (настройками).

4. Когда тесты терпят неудачу, они должны предоставить как можно больше информации о проблеме. googletest не останавливается на первом провале теста. Он останавливает только текущий тест и переходит к следующему. Вы также можете настроить тесты, которые сообщают о некритических сбоях, после которых текущий тест продолжается. Таким образом, вы можете обнаруживать и исправлять несколько ошибок за один цикл запуск-редактирование-компиляция.

5. googletest автоматически отслеживает все заданные тесты (defined tests) и не требует от пользователя их перечисления для их запуска.

6. Тесты должны быть быстрыми. С помощью googletest можно повторно использовать общие ресурсы в тестах и «платить» за создание/разрушение тестовой среды только один раз (set-up /tear-down), не делая тесты зависимыми друг от друга.

Поскольку googletest основан на популярной архитектуре xUnit, его легко освоить, если раньше использовали JUnit или PyUnit. В противном случае вам потребуется немного времени, чтобы изучить основы и приступить к работе.

===== **** =====