Занятие ESDP #1 - Основы приемочного тестирования, Issue Tracker

Тестирование ПО

В процессе разработки ПО очень важную роль необходимо отвести тестированию, как одному из самых успешных методов контроля качества ПО. Под качеством понимается, что продукт в результате:

- удовлетворяет всем требованиям заказчика
- с помощью продукта решаются определенные задачи
- при использовании продукта заказчик не несет потерь, связанных с неверной реализацией (баги, ошибки, недостаточная защита)

При ручном тестировании всех сценариев использования приложения возникает проблема, связанная с человеческим фактором - при каждом внесении изменения в приложение, человек должен будет проверить весь уже существующий функционал для того, чтобы убедиться, что все работает по-прежнему правильно. Чем больше становится возможных вариантов использования (сценариев), тем больше времени человек тратит на полную проверку приложения. Таким образом, лучше всего использовать **автоматизированное тестирование**.

Под автоматизированным тестированием можно понять написание другой (часто отдельной) программы, которая будет проверять функции целевой программы.

Чаще всего используются следующие виды автоматизированного тестирования:

- Модульное тестирование (Unit testing)
- Приемочное тестирование (Acceptance testing)

Модульное тестирование - это тестирование, которое проверяет систему "по частям", проверяя отдельные классы и их методы так, чтобы они возвращали правильные значения по заданным входным данным (таблице примеров). При этом правильное модульное тестирование обычно как можно сильнее отделяет все зависимости модуля, так чтобы проверять работу только самого модуля, без его возможных зависимостей.

Приемочное тестирование - это тестирование, которое проверяет функционал приложения с точки зрения его использования конечным пользователем (иногда даже самим заказчиком). Этот способ тестирования чаще всего проверяет систему через пользовательский интерфейс приложения, и что заданные сценарии использования приложения приводят к правильным результатам.

Также существуют другие виды тестирования, которые могут быть автоматизированными, или нет, но они проверяют другие аспекты приложения, помимо основного функционала, например:

- Тестирование удобства использования
- Тестирование пакета установки
- Тестирование (репетиция) поставки
- Нагрузочное тестирование (Load testing, performance testing)
- и т.п.

Таким образом, написание автоматизированных тестов облегчает процесс проверки качества приложения - теперь вместо человека приложение будет проверять другая программа, которая сделает это быстрее и зачастую правильнее, т.к. исключается человеческий фактор.

При разработке приложения с использованием автоматизированного тестирования часто применяется методика TDD (test-driven development), или "разработка через тестирование", при которой сначала пишется тест, а затем сам функционал. При этом часто используется такая схема наименования - если это модульный тест, то подход называется TDD, если это приемочный тест, то - BDD (behavior-driven development).

Приемочное тестирование

При приемочном тестировании тесты обычно состоят из двух частей - описания сценариев использования и самих шагов, необходимых для тестирования. При этом сценарии использования часто пишутся на "человеческом" языке, например, английском или русском.

Одним из таких специальных языков для описания сценариев использования при приемочном тестировании является язык Геркин (Gherkin), который описывает сценарии с помощью некоторых ключевых слов, например:

```
Функция: Короткое, но исчерпывающее описание требуемого функционала
    Для того, чтобы достичь определенных целей
    В качестве определенного участника взаимодействия с системой
    Я хочу получить определенную пользу

    Сценарий: Какая-то определенная бизнес-ситуация
        Допустим какое-то условие
        И ещё одно условие
        Если предпринимается какое-то действие участником
        И им делается ещё что-то
        И вдобавок он совершил что-то ещё
        То получается какой-то проверяемый результат
        И что-то ещё случается, что мы можем проверить
```

Ключевые слова в данном случае выделены жирным шрифтом. Далее обработчик разбивает файл с таким тестом на функции, сценарии и входящие в них шаги. Давайте разберем этот пример:

- Строка **Функция**: Короткое, но исчерпывающее описание требуемого функционала начинает собой описание функционала и дает ему название.
- Следующие три строчки не обрабатываются и не несут никакой смысловой нагрузки для обработчика тестов, но они задают контекст тестирования и одновременно описывают, какую пользу мы получим от этого функционала.
- Строка **Сценарий**: Какая-то определенная бизнес-ситуация начинает сценарий и содержит его описание.
- Следующие 7 строк описывают шаги теста, каждому из которых впоследствии будет сопоставлен определенный программный код, выполняющий описанное действие. Сопоставлению подлежат части строк лежащие после ключевых слов “Допустим”, “И”, “Если” “То” и т.д.

Считается хорошим тоном, когда сценарий пишется кратко и состоит из трех основных строк:

- **Допустим** (или **дано**)
- **Если** (или **когда**)
- **То** (или **тогда**)

Один из самых популярных обработчиков языка Gherkin (или мы его можем в дальнейшем называть **тестовым фреймворком**) является пакет cucumber.

Для включения в ASP.NET проект приемочных тестов используется библиотека SpecFlow, информацию о которой можно найти здесь:
http://specflow.org/getting-started/
Подробнее данный тип тестирования мы пройдем на следующих лекциях.

Issue tracker

В процессе разработки необходимо всегда обеспечивать информацию - кто в команде чем занимается, а также отслеживать список задач, которые еще осталось сделать. Эти задачи и многие другие для управления проектами решает сорт программного обеспечения, которое называется "Системы управления задачами", или "Issue trackers".

Для разработчика Issue Tracker - это в первую очередь, место, где хранятся все задачи, которые нужно выполнить ему или его команде для достижения результата. Нередко задачи заполняются также и самим разработчиком, но тут все зависит от процесса.

Каковы основные случаи использования Issue Tracker?

1. Вы можете распланировать весь проект и разбить его на задачи. Все задачи можно записать в issue-tracker. Потом, по мере реализации, помечать задачи как сделанные.
2. Используя инструмент с поддержкой трекинга времени вы можете записывать, сколько времени вы затратили на каждую задачу и таким образом, можно понять вашу скорость работы (или скорость вашей команды) для более эффективного управления временем и ожиданиями заказчика.
3. Если делать коммиты с указанием номера задачи, то при code review или просто, разбираясь в старом коде, можно точно сказать, зачем написана та или иная строка кода.
4. Подробное описание задачи и причин ее возникновения позволяет вести грамотную документацию проекта. Помните - самая важная документация проекта - это его исходный код. Если можно понять, какую задачу он решает, и прочитать описание тикета, по которому он написан, можно легко войти в контекст решаемой задачи.

Задача в системах контроля также называется "тикетом" ("ticket"). У тикета традиционно есть такие поля:

- **Тип задачи**. Обычно бывает одним из:
    - task (задача)
    - defect или bug (дефект, "баг")
    - enhancement (улучшение)
- **Summary (или "заголовок")** - краткое описание задачи. Не должно быть слишком абстрактным, но и не слишком длинным. Должно обязательно сразу отражать общую суть задачи.
    - Примеры хороших Summary:
        - Сделать систему аутентификации (для *задачи*)
        - Пользователь автоматически не входит в систему после аутентификации (для *бага*)
        - Улучшить оформление подсказок по ошибкам аутентификации (для *улучшения*)
    - Примеры плохих Summary:
        - asdfqwerty
        - Сделать задачу 40 из эксельки
        - Пользователь не может войти в систему
- **Description (описание)** - полное описание задачи. Для разных типов задач может отличаться. Например,
    - Для дефекта нужно отражать следующую информацию:
        - шаги воспроизведения
        - наблюдаемое поведение
        - ожидаемое поведение
        - экземпляр (версия ПО, окружение) котором был обнаружен дефект
        - сценарий тестирования
    - Для задачи:

        - Конечный результат
        - План решения (Что нужно сделать, Как это сделать)
        - Мотивация (Зачем это нужно потребителю)
        - Критерии приемки
    - Для улучшения:
        - Проблема в наблюдаемых явлениях
        - Факты неудобства при использовании функции системы
        - Предлагаемое решение проблемы.
- **Status (статус)** - текущее состояние задачи. Только что созданная задача получает как правило статус new. После взятия задачи в работу, она может получить статус "in progress", "accepted" и т.п. После завершения задачи, она получает статус "finished", "closed". Также бывают исключительные статусы состояний, например "on hold", "paused", "idle" - приостановлена, "worksforme" - отклонена по причине невозможности воспроизведения, и т.п. В основном статусы и процесс перехода из одного статуса в другой - назначаются в зависимости от технологического процесса, принятого на предприятии или в конкретном проекте.

Также после создания тикета ему назначается **уникальный идентификатор** в системе, которую вы используете. Как правило, такой идентификатор выглядит как порядковое число (#56), но также бывают и такие идентификаторы: PROJECT-123, где PROJECT - код проекта в системе.

После того, как вы получаете тикет "в работу", вы должны пометить его как "принятый", или иначе говоря, установить его **статус** в новое состояние. Практически во всех системах контроля задач при этом вы помечаетесь как исполнитель данной задачи. Также администраторы или менеджеры проекта могут назначать людей на выполнение задач.

При выполнении работ по задаче, вы делаете коммиты с пометкой идентификатора вашего тикета. Например,

```
#56 - Added authentication.
```

Таким образом, номер тикета лучше всего включать в первую строку описания коммита, и как правило пишется в самом начале этого сообщения.

Многие системы контроля задач умеют подключаться к вашему удаленному репозиторию и автоматически ведут учет всех коммитов по данной задаче и ведут некий "дневник" того, что происходит с задачей:

- Когда она была создана
- Когда в ней произведены изменения
- Когда она переведена из статуса в статус
- Когда по ней сделаны коммиты и кем, обычно со ссылкой на просмотр коммита.

Все это облегчает управление проектом, в котором вы находитесь, поэтому необходимо правильное выполнение процесса предприятия всеми участниками проекта.