4.17.Разработка через тестирование. Типичный цикл разработки через тестирование. Преимущества и недостатки разработки через тестирование. Инструменты, поддерживающие разработку через тестирование.

Разработка через тестирование — Test-Driven Development (TDD) — техника программирования, при которой модульные тесты для программы или ее фрагмента пишутся **до** самой программы и, по существу, управляют ее разработкой. Является одной из основных практик экстремального программирования.

Разработка в стиле TDD состоит из коротких циклов (длительностью от 2 минут, в зависимости от опытности и стиля работы программиста). Каждый цикл состоит из следующих шагов:

1. Из репозитория извлекается программная система, находящаяся в согласованном состоянии, когда весь набор модульных тестов выполняется успешно.
2. Добавляется новый тест. Он может состоять в проверке, реализует ли система некоторое новое поведение или содержит ли некоторую ошибку, о которой недавно стало известно.
3. Успешно выполняется весь набор тестов, кроме нового теста, который выполняется неуспешно. Этот шаг необходим для проверки самого теста -- включен ли он в общую систему тестирования и правильно ли отражает новое требование к системе, которому она, естественно, еще не удовлетворяет.
4. Программа изменяется с тем, чтобы **как можно скорее** выполнялись все тесты. Нужно добавить самое простое решение, удовлетворяющее новому тесту, и одновременно с этим не испортить существующие тесты. Большая часть нежелательных побочных и отдаленных эффектов от вносимых в программу изменений отслеживается именно на этом этапе, с помощью достаточно полного набора тестов.
5. Весь набор тестов выполняется успешно.
6. Теперь, когда требуемая в этом цикле функциональность достигнута самым простым способом, программа рефакторится для улучшения структуры и устранения избыточного, дублированного кода.
7. Весь набор тестов выполняется успешно.
8. Комплект изменений, сделанных в этом цикле в тестах и программе коммитится в репозиторий, после чего программа снова находится в согласованном состоянии и содержит четко осязаемое улучшение по сравнению с предыдущим состоянием.

Этот цикл упрощенно называется "красный-зеленый-рефакторинг". Красный и зеленый — это цвета полоски в среде тестирования JUnit, которая показывает все тесты сработали или нет. При этом на первом ("красном") этапе необходимо добиться того, чтобы программа просто компилировалась, без срабатывания добавленного теста.

Сравним классический процесс разработки софта и TDD:

4.17

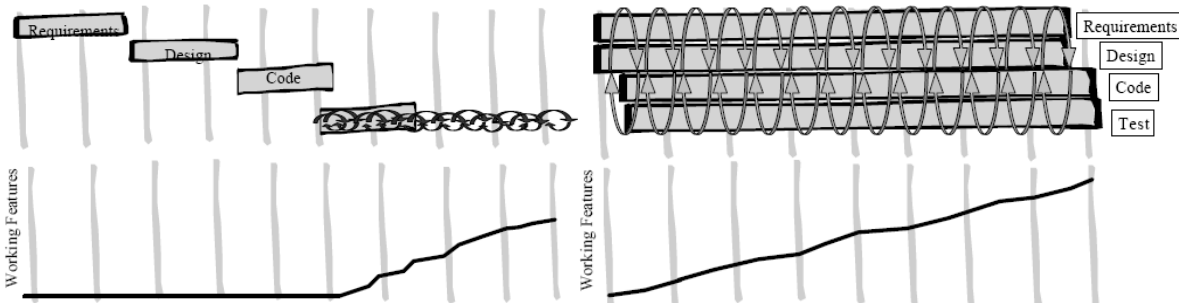
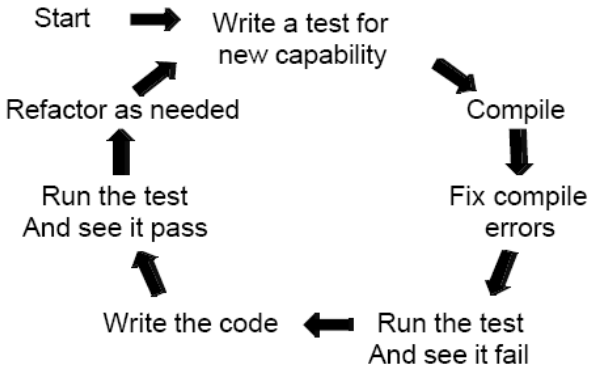


График показывает число работающих фич, в зависимости от времени (в месяцах). При TDD-шном подходе фичи появляются с первых же дней работы, при классическом — работающая прога появляется через полгода после начала проекта.

Еще о правилах TDD:



Всегда выбирай самое простое решение! "Keep It Simple, Stupid" (KISS)

- Рассматривай простейший подход, который теоретически может сработать;
- Пиши самый простой код, который позволит пройти все тесты;
- Рефактори код до самого простого дизайна;
- Убей все повторяемые куски кода.

Автоматическое тестирование

- Юнит-тесты
  - Показывают, что код делает то, что программист от него ожидает;
  - Являются примерами использования классов и методов, то есть несут функцию документации;
  - Запускаются каждые несколько минут;
  - Правила:
    - Каждый класс/модуль имеет хотя бы один юнит-тест;
    - Тестируется всё, что теоретически может упасть;
    - Если теоретически не может упасть, не тестируй это.
- Acceptance tests (тесты, проверяющие работу софта в целом)
  - Их можно показывать заказчику, чтобы доказать, что такая-то фича заимплеменчена;
  - Запускаются ежедневно или чаще.
- Все тесты всегда автоматические.

Фреймворки

- jUnit, CppUnit, PyUnit, DUnit NUnit, VBUnit, есть везде;
- Собирают, огранизовывают и автоматически запускают ваш код;
- Можно использовать графические запускатели тестов

    - Зелёная полоска (всё хорошо) поднимает настроение программисту, это правильно (сравни с зелёной надписью Accepted в ACM);
- Запустить тесты должно быть легко и просто, как и сбилдить.

Достоинства

- Кодится лучше и быстрее;
- Не думаешь о дизайне — он получается сам;
- Про каждый кусок кода известно, и даже есть пример (юнит-тест), как он работает;
- Весь код всегда покрыт тестами — высокий уровень доверия к старому коду;
- Серьёзный баг (требующий мегарефакторинга) с большей вероятностью будет обнаружен на ранней стадии;
- Код получается модульный, гибкий и расширяемый.

Недостатки

- Плохо применим, когда задействованы ресурсы: база данных, инет и т. д.
- Хорошо TDD-шничать — сложное искусство, подвластное не всем кодерам;