SolarLab>_

Unit testing in .net

Алексей Андреев https://t.me/AlekseyAndreev1984

Содержание

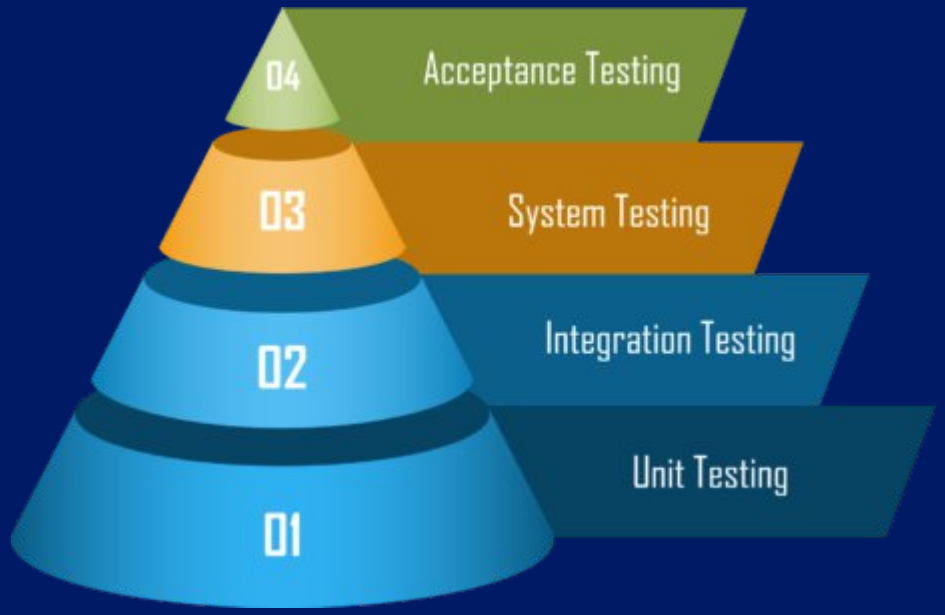
SolarLab>_

Введение

Где в пирамиде находятся юнит тесты?

Многие слышали про пирамиды тестирования. Давайте поймём, где в этих пирамидах находятся юнит тесты

Пирамида 4х уровневая

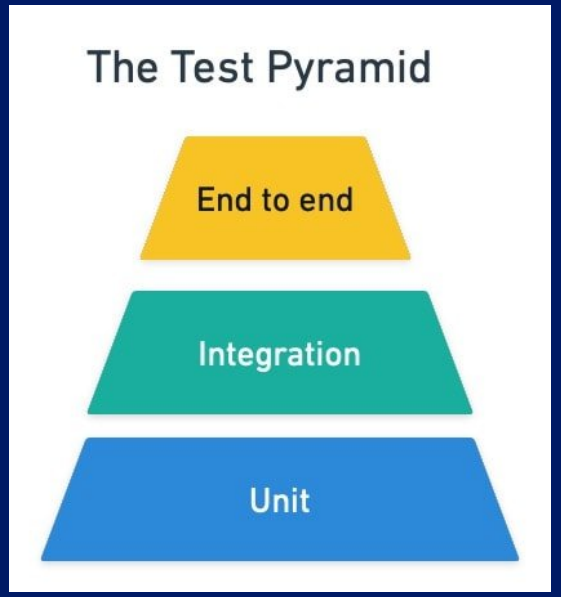


Пирамида тестирования, также часто говорят уровни тестирования, это **группировка тестов по уровню детализации и их назначению.** Эту абстракцию придумал Майк Кон и описал в книге «Scrum: гибкая разработка ПО»

Пирамиду разбивают на **4 уровня**:

- модульное тестирование (юнит);
- интеграционное тестирование;
- системное тестирования;
- приемочное тестирование.

Пирамида 3х уровневая



Но **можно встретить** варианты, где **3 уровня**. В этой модели объединяют интеграционный и системный уровни:

- модульное тестирование (юнит);
- интеграционное тестирование;
- приемочное тестирование.

Понятия в юнит тестировании

Юнит-тестирование - тестирование одного продакшн юнита в полностью контролируемом окружении.

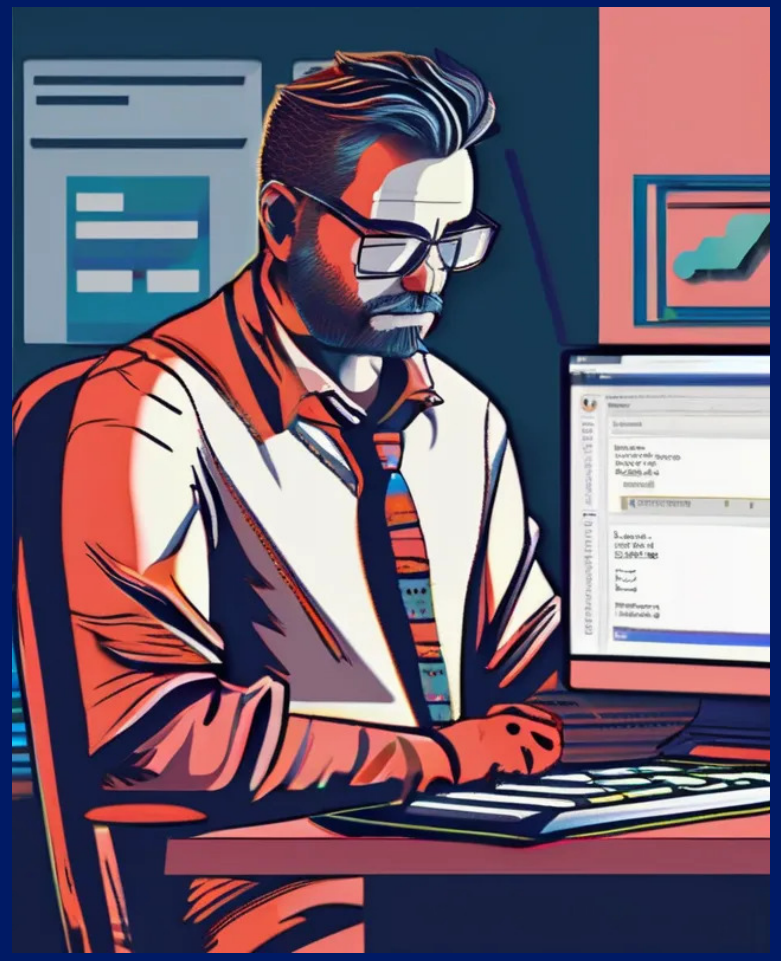
Продакшн юнит - обычно класс, но может быть и функция, и файл. Важно, чтобы юнит соответствовал принципал SOLID, в этом случае юнит-тесты будут лаконичными. Юниты узкоспециализированы и очень хорошо выполняют одну конкретную задачу, для которой они созданы. В большинстве случаев юниты взаимодействуют друг с другом, делегируя выполнение специализированных задач.

Полностью контролируемое окружение - окружение, имитирующие среду, внутри которой юнит работает как в реальном приложении, но полностью открытое для тестирования и настройки. Поведение окружения задается для конкретного тест-кейса через лаконичный API и любое поведение вне этого кейса для него не определено. В этом окружении не должно быть других продакшн юнитов, иначе возрастает сложность тестов и из юнит-тестирования мы переходим к интеграционному тестированию.

SolarLab>_

Зачем нужны юнит тесты?

Swagger



SoapUI

POSTMAN

APACHE JMeter™

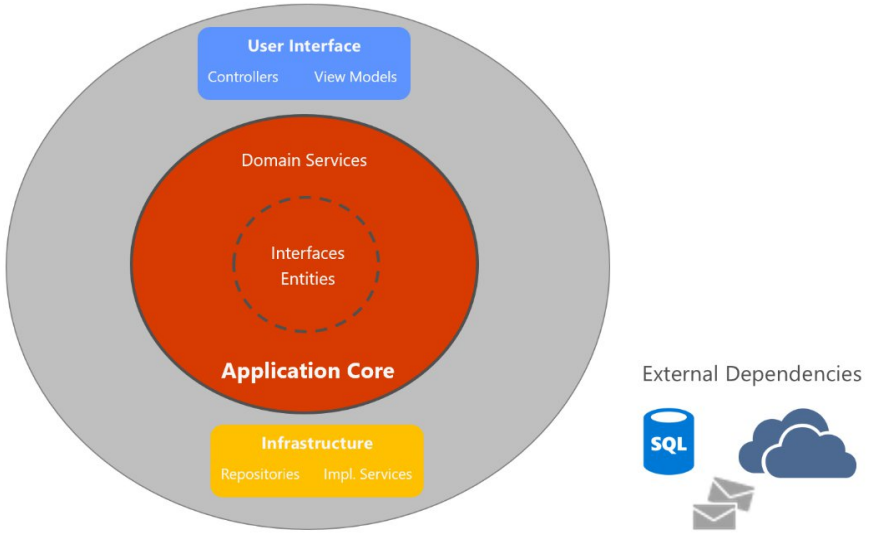
Зачем нужны юнит тесты?

- При качественном покрытии юнит тестами, если разработчик поменяет логику приложения – юнит тесты упадут и покажут изменения. Либо у нас неверно изменена логика(правим код), либо новые требования(правим юнит тесты)
- Юнит тесты могут(и должны) выполняться во время деплоя приложения, что даёт небольшую гарантию, что приложение не сломалось
- Чтобы не выполнять проверки кода путём запуска своего приложения, или нескольких приложений. А проверить нужный участок сразу и быстро

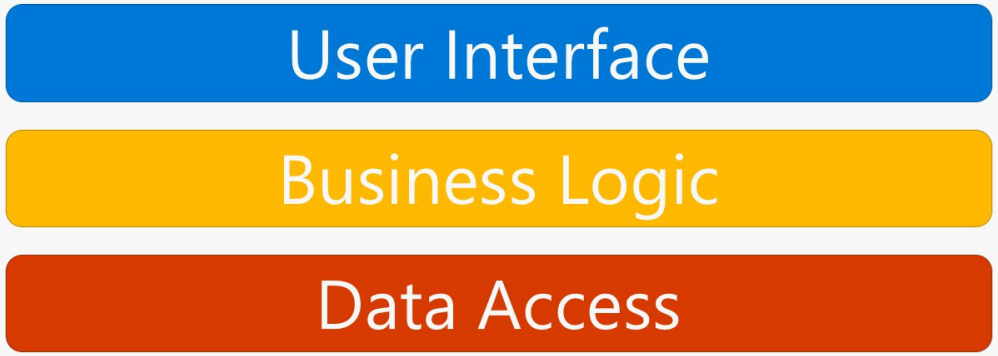
SolarLab>_

Что мы покрываем юнит тестами?

Clean Architecture Layers (Onion view)



Application Layers



Что мы покрываем юнит тестами?

Архитектура бывает разная

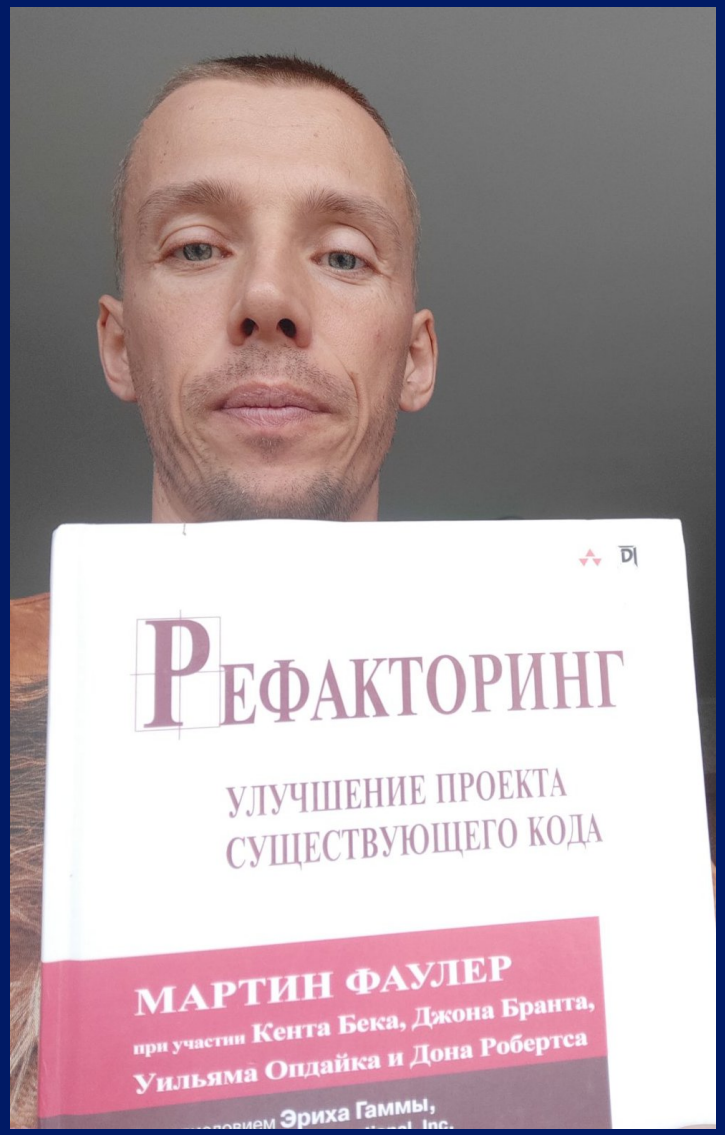
В чистой архитектуре, или в архитектуре трёх(четырёх или пяти) слойной чётко выделяется слой бизнес логики.

И покрывать в первую очередь стоит всю бизнес логику юнит тестами.

В дальнейшем все части приложения должны быть покрыты юнит тестами

SolarLab>_

Юнит тестирование и рефакторинг



Рефакторинг и юнит тесты

Не стоит начинать рефакторинг, если у вас нет юнит тестов(вольный перевод по книге Мартина Фаулера Рефакторинг)

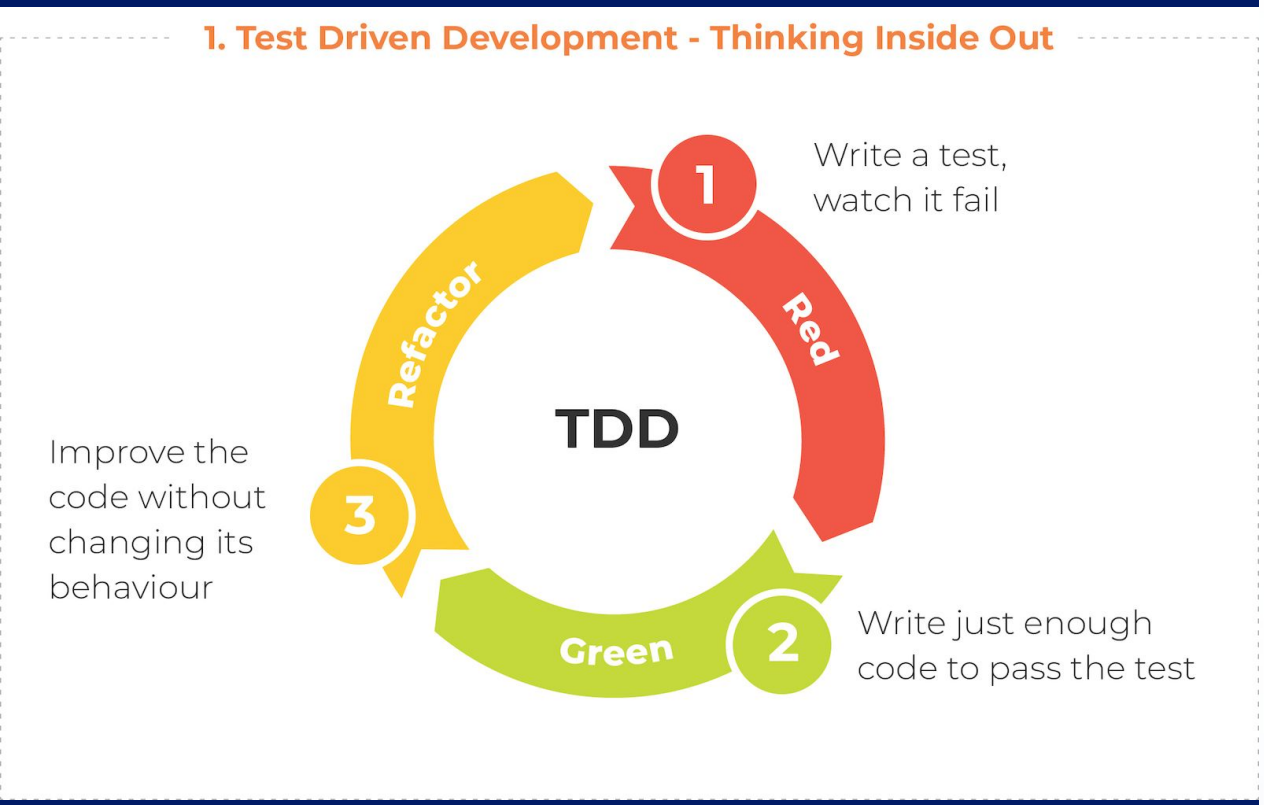
Первый шаг рефакторинга

Приступая к рефакторингу, я всегда начинаю с одного и того же: строю надежный набор тестов для перерабатываемой части кода. Тесты важны потому, что, даже последовательно выполняя рефакторинг, необходимо исключить появление ошибок. Ведь я, как и всякий человек, могу ошибиться. Поэтому мне нужны надежные тесты.

SolarLab>_

TDD - является ли необходимостью?

TDD



Вот основные принципы применения TDD:

- Прежде чем писать код реализации некоей возможности, пишут тест, который позволяет проверить, работает ли этот будущий код реализации, или нет. Прежде чем переходить к следующему шагу, тест запускают и убеждаются в том, что он выдаёт ошибку. Благодаря этому можно быть уверенным в том, что тест не выдаёт ложноположительные результаты, это — своего рода тестирование самих тестов.
- Создают реализацию возможности и добиваются того, чтобы она успешно прошла тестирование.
- Выполняют, если это нужно, рефакторинг кода. Рефакторинг, при наличии теста, который способен указать разработчику на правильность или неправильность работы системы, вселяет в разработчика уверенность в его действиях.

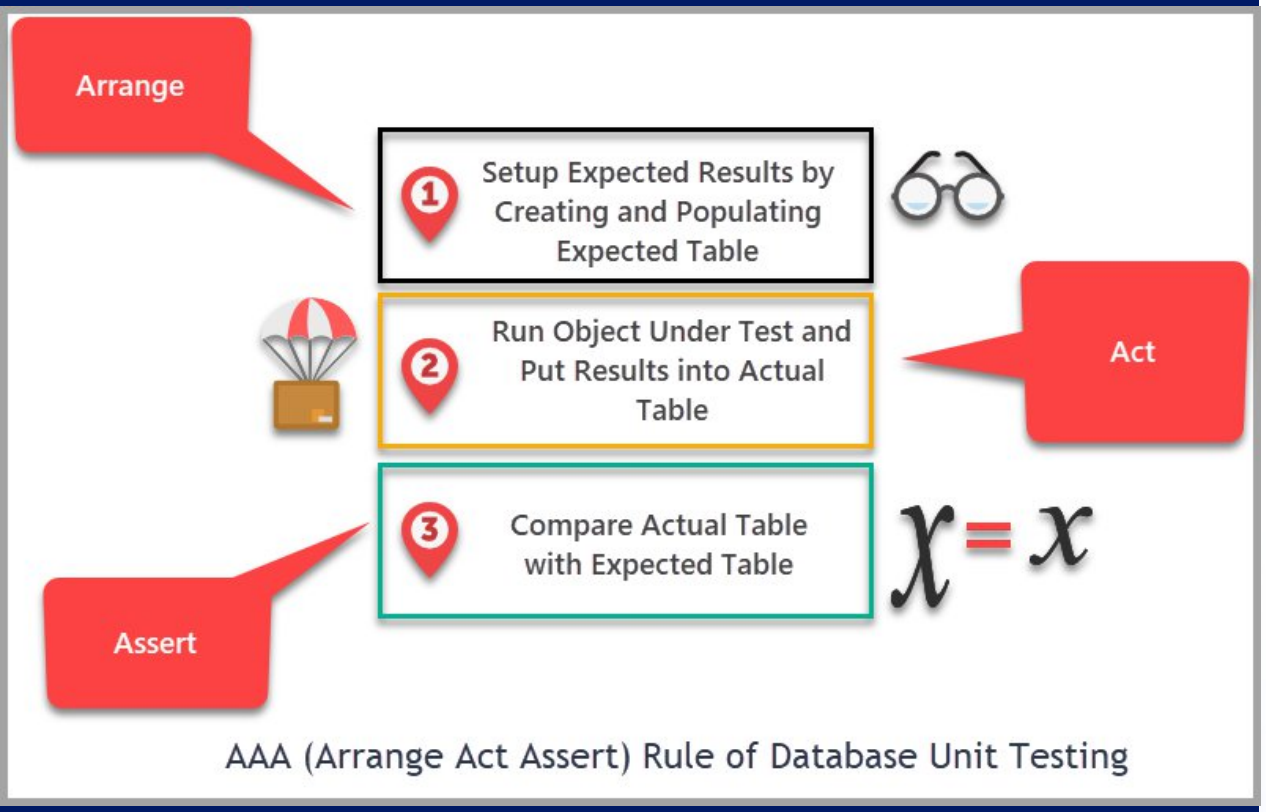
Недостатки TDD:

- медленный процесс(спорно)
- Все члены команды должны это делать и понимать как это делать
- Тесты должны поддерживаться при изменении требований(тяжело для проектов с часто меняющимися требованиями)

SolarLab>_

AAA паттерн

AAA – Андреев Алексей Александрович? Или Arrange Act Assert



Arrange — в этом блоке кода мы настраиваем тестовое окружение тестируемого юнита;
Act — выполнение или вызов тестируемого сценария;
Assert — проверка того, что тестируемый вызов ведет себя определенным образом.

Этот паттерн улучшает структуру кода и его читабельность

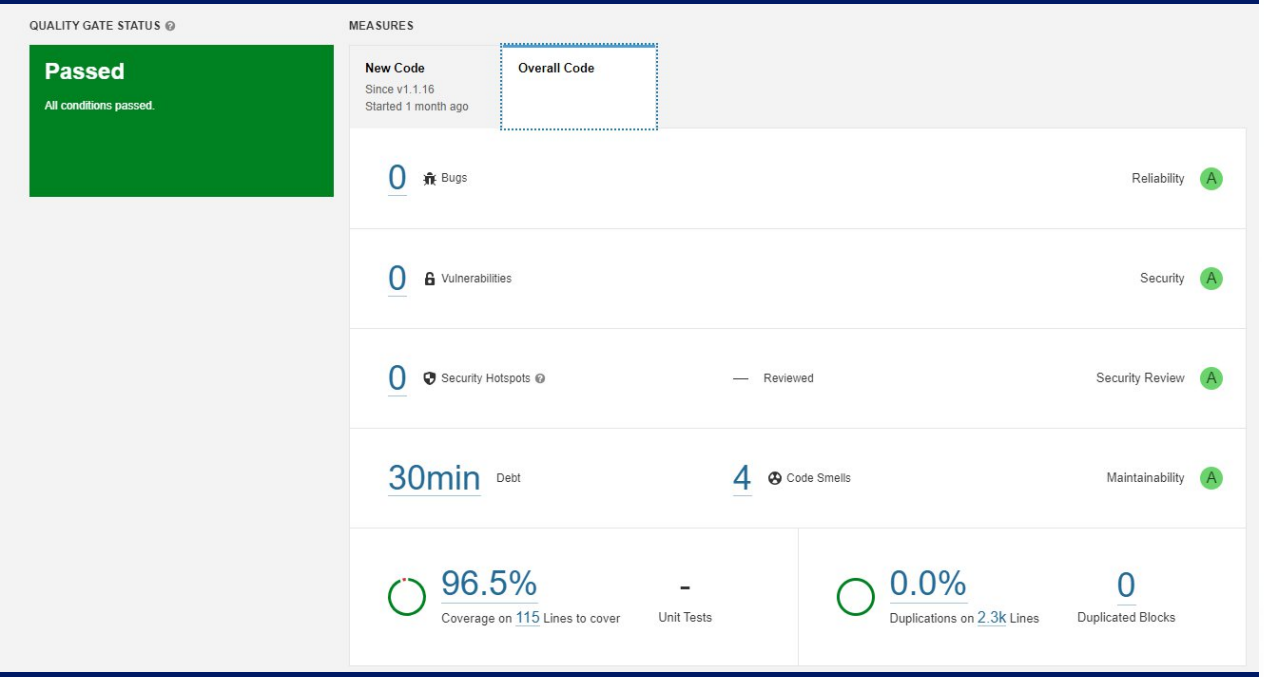
Рассмотри реальный проект с примерами

Вместе на такси

https://togetherbytaxi.ru

SolarLab>_

Количество покрытия кода юнит тестами - codecoverage



codecoverage minicover

https://github.com/lucaslorentz/minicover

dotnet restore

dotnet build

minicover instrument

minicover reset

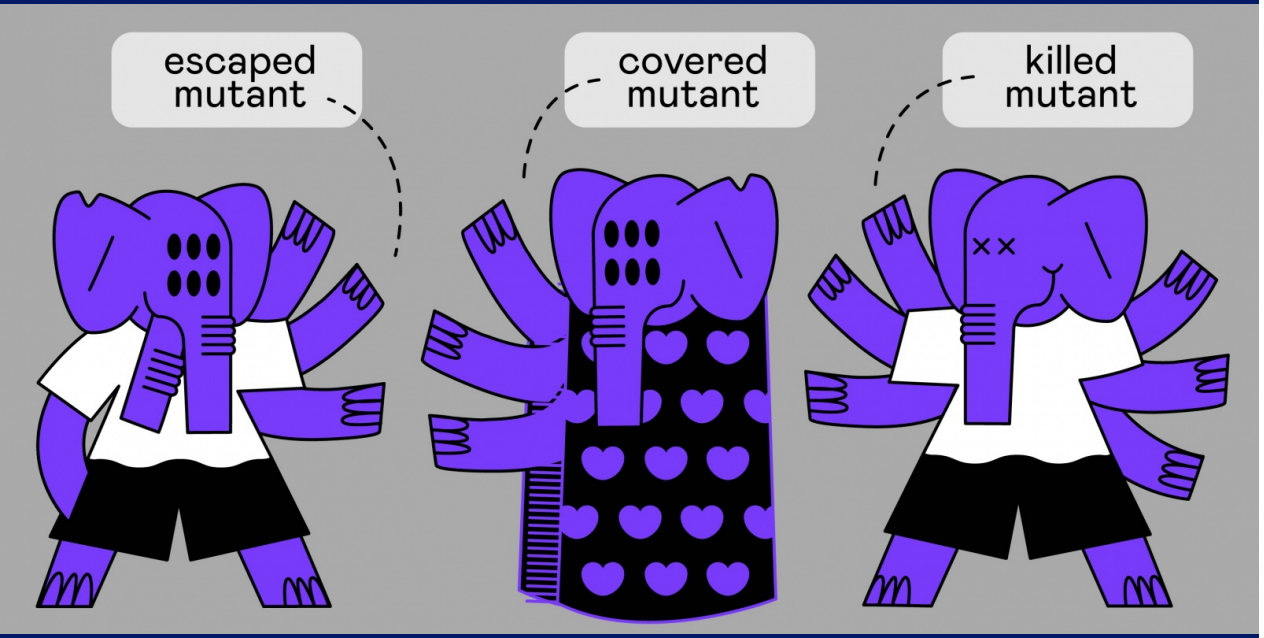
dotnet test --no-build

minicover uninstrument

minicover htmlreport --threshold 90

SolarLab>_

Качество покрытия кода юнит тестами - мутационные тесты Stryker



Мутационные тесты Stryker

https://stryker-mutator.io/docs/stryker-net/getting-started/

dotnet tool install -g dotnet-stryker

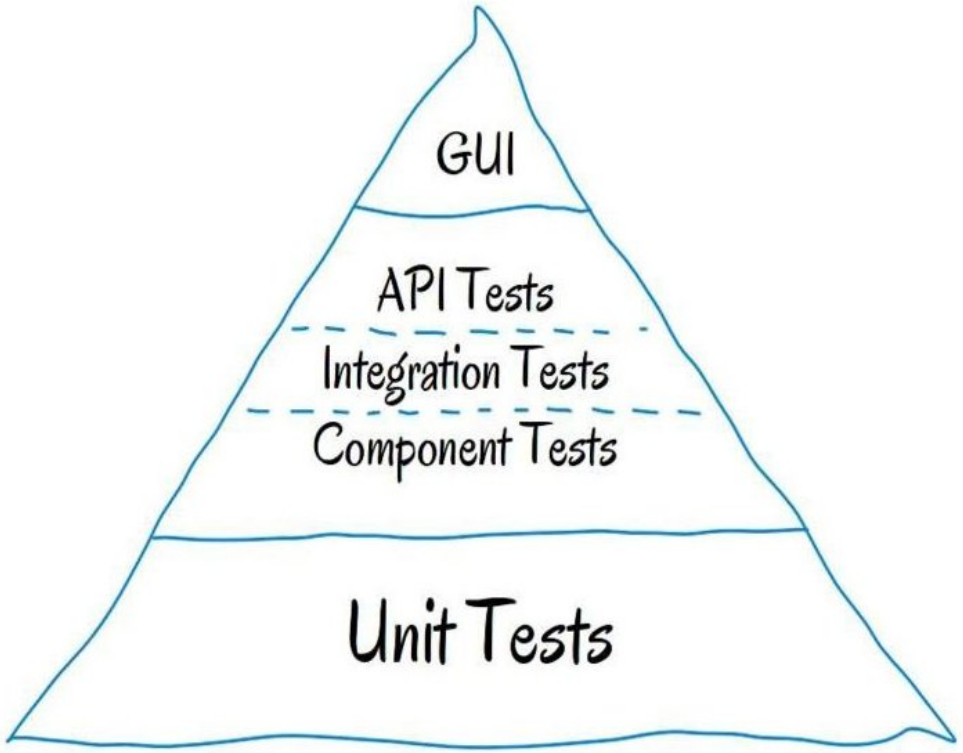
dotnet stryker

SolarLab>_

Не стоит зацикливаться на юнит тестах

Юнит тесты помогают,но не являются панацеей или серебряной пулей

Unit тестов недостаточно



- Не зацикливайтесь на unit тестах
- В унаследованных системах начинайте сверху

- Юнит тесты могут на старте проекта(когда делаем MVP) сильно увеличить время на разработку, что для быстрых прототипов не допустимо

- Слишком высокое покрытие может привести к раздуванию тестового набора и усложнению его поддержки.

SolarLab>_

Выводы

Выводы

- Юнит тесты важно писать каждому разработчику. Это ускоряет процесс проверки текущего функционала, помогает избежать возможных будущих ошибок в коде.
- Юнит тесты могут на старте проекта(когда делаем MVP) сильно увеличить время на разработку, что для быстрых прототипов не допустимо
- Для качества покрытия используйте готовые инструменты(codecoverage, stryker)

SolarLab>_

Ссылки

Ссылки

- **Майк Кон: Scrum: гибкая разработка ПО** Succeeding with Agile: Software Development Using Scrum (Addison-Wesley Signature Series (Cohn)) 1, Cohn, Mike, eBook - Amazon.com
- Подробнее про пирамиду тестирования / Хабр (habr.com)
- Анатомия юнит тестирования / Хабр (habr.com)
- Clean Coder Blog
- Почему юнит тесты не работают. История большого проекта (Александр Мартюшев, AgileDays-2015) — 0x1.tv
- Рефакторинг - улучшение существующего кода (avmim.com)
- GitHub - microsoft/codecoverage
- GitHub - lucaslorentz/minicover: Cross platform code coverage tool for .NET Core
- Getting started | Stryker Mutator (stryker-mutator.io)
- Aleksey Andreev / TogetherByTaxi · GitLab
- https://togetherbytaxi.ru/

Вопросы?

SolarLab>_

Спасибо за внимание