Development III – Unit testing, good practices

Štěpán Kozák

Kentico

Revision from Last Lesson

- What is the performance testing good for?
- What is the difference between unit tests and integration tests?
- **Why the hell do we spend second lecture on testing?**

Unit vs. Integration Tests

- **Unit testing** - basic testing of small unit functionality
  - NO external dependencies
  - NO need for configuration
  - NO influence on other tests running in parallel
  - NO change in results no matter how many times we run the test

- **Integration testing**
  - May have external dependencies
  - May need some configuration
  - May have influence on other tests running in parallel

Why do We Write Automatic tests?

- **What are the benefits** of having testable code covered with (unit|integration) tests?
- Did it ever happen to you?
  - You build your code on a API created by your colleague, everything was working, you made no changes to your code and suddenly your code does not work anymore
- **What are the disadvantages** of the approach?

So … Why then?

- Anytime somebody **changes public interface** of a code you rely on, the bug is found immediately
- It **forces** you **to write a „better“ code** following well-established patterns
- **Speeds up the testing process** in case of huge applications (backward compatibility, new functionality)
- **Speeds up debugging process** (you can identify the source of a bug faster)

What about the Disadvantages?

- Takes some time to write (good) tests
- Takes (significant) amount of time learn how to write (good tests)
- Delusion of having no bugs and completely correct code when all tests pass

Good Practices When Writing Tests

- Confidentiality (any time you make a modification to a code, **ALL the tests have to pass**, no exceptions)

- **No complex expressions** (no if-else statements, no try-catch, …)
  - Test cannot contain any bug ☺
  - Test has to be readable and managable by anybody without the deep knowledge of given functionality

Some Questions about Automatic Tests

- Who creates the unit tests?
- In which phase of the development process are the tests created?
- Can I change the code of a test when I don‘t succeed in having it passed?

Test Driven Development

1. **Test creation** at the beginning for ALL the functionality intended for production
2. **Implementation** of given functionality so that all the tests pass
3. **Improvements** of production code (refactoring)

Main rule of TDD:

- Production code cannot be modified until there is a test which proves its incorrectness

Code Coverage

- Have it as close to 100% as possible, however …
- **100% code coverage does not prove 100% correctness!**

How to Fake the Tests …

- Never fake tests! ☺
- Fake objects you use in the tests to get rid of the external dependencies and configuration

- The technique is called **mocking**
  - Manual mocks
  - Automatic mocks (frameworks, e.g. NSubstitute)

Testable Code

- When is it easy to write (real) unit tests for a class?
- **Dependency injection**
  - „Is a software design pattern that allows a choice of component to be made at run-time rather than compile time. This can be used, for example, as a simple way to load plugins dynamically or to choose mock objects in test environments vs. real objects in production environments“