Testing and Security

6. maj 2018

Agenda

Testing

Security

Testing

Why test?

Software contains defects, they hide themselves

Defects can cause software failures

Failures can cost money.

- Even mortal…

Testing assures the quality of the software

Testing accelerates software development

Is debugging considered testing?

Famous software failures

Ariane-5

- Took navigation software from Ariane-4, did not test.
- Everything else was thoroughly tested
- Ariane-5 had a different trajectory
- When converting from 64bit float to 16bit unsigned int, got arithmetic overflow ☹
- There was a exception handler, it was disabled ☹

Heartbleed

- Heartbleed is an extension of TLS
- It is used to keep Datagram TLS sessions open
- Simple request/response scheme:
  - *“Send me back the following (padded) string which is **n** bytes long…”*
- An attacker just had to request a long string, while telling it is short
- Other party responded with a long string and potentially leaked confidential data

Do testing right

Testing has to be planned

Testing costs easily up to 40% of a projects budget

Testing has to be performed in a reasonable way

Testing shall be independent and objective

Testing has to be managed

Software testing is a fundamental part of professional software development!

What is Testing

*The process consisting of all life-cycle activities, both static and dynamic, concerned with **planning**, **preparation** and **evaluation** of software products and related work products to determine that they **satisfy specified requirements**, to demonstrate that they are fit for purposes and to **detect defects***

What is testing (cont.)

Perspective	Objective
Developer	Find defects
Quality Assurance	Evaluate quality
Tester	Check functionality against requirements
Customer	Verify usability and applicability

What is requirements and specification?

The **requirements** document is the input to a development phase

The **specification** document is the output of a development phase

The specification document resulting from development phase X serves a requirements document for development phase X+1

Requirements and specification are the same, depending on the usage and point of view

The product is 100% specification

Validation and Verification

Validation

The process of evaluating a system or component during or at the end of the development process to determine whether it ***satisfies specified requirements***

- IEEE Std. 610.10-1990

“Are we building the right product?”

Verification

The process of evaluating a system or component to determine whether the system of a given development phase ***satisfies the conditions imposed*** *at the start of that phase*

- IEEE Std. 610.10-1990

“Are we building the product right?”

Static testing

Static testing

Verification only

Never executes code

Find defects

Static analysis

- Automated

Reviews

- Manual examination
- Too support

Statis analysis

Analyse code

- Control flow
- Data flow

Typical usecases

- Syntax checking
- Code smell detection
- Coding standards compliance

Typical detected defects

- Syntax violations
- Unreachable code
- Overly complicated constructs

Review

Different review types

- Informal review
  - No formal process and therefore inexpensive
- Walkthrough
  - Train colleagues and users, gain common understanding
- Technical review
  - Documented, defined process, allows discussion
- Inspection
  - Formal process, gain metrics

Review (cont.)

Success factors

- Clear predefined objective
- Defects found are welcomed and expressed objectively
- Application of suitable review techniques
- Management supports review process
- Emphasis on learning and process improvement

Dynamic testing

Dynamic testing

Primarily verification

Executes code

Find failures

Different testing methods

- Black box testing
- White box testing

Different test levels

- Unit (component, module) testing
- Integration testing
- System testing
- Acceptance testing

Dynamic testing methods

Black-box testing

Specification-based testing

Test functionality by observing external behavior

No knowledge of internal workings

- This is required!

Different approaches

- Equivalence partitioning
- Boundary value analysis
- Decision table testing

White-box testing

Structure-based testing

Close examination of procedural level of detail

Knowledge of internals is required

Different approaches

- Statement testing
- Decision testing
- (Multiple) Condition testing

Test levels

Unit Testing

Test individual units in isolation

Find defects in software components

- e.g. functions, methods and classes

Done by developers

- During development phase
- Hopefully before implementing a given feature

This is considered white-box testing

Integration Testing

Test communication/interaction of units

- I.e. their interfaces

Possibly separate unit integration and system integration tests

Done by developers

Primarily white-box tests, but black-box can also be done

System Testing

Test a complete integrated system

Evaluate system compliance with requirements

Stress, performance, usability, etc testing

Done by testers

- Preferably external

In general black-box tests

- White-box tests is possible but not common

Acceptance Testing

Test a complete integrated system

Evaluate system compliance with acceptance criteria

May be performed during development phase

Done by customers/users

Strictly black-box tests

Continuous Integration

Automated activities

- Execute static code analysis
- Execute unit tests and maybe check code coverage
- Deploy to test environment
- Execute integration tests

Benefits

- Earlier detection and analysis of conflicting changes
- Regular feedback on whether the code is working as intended
- No big-bang deploy
- Reduces repetitive manual testing activities

Testing in .NET

Unit testing in .NET Core

Create separate testing project

For each class in project, have a test class associated

In test classes create test methods for validating behavior in project

Each test method is based around *Arrange, Act* and *Assert*

Arrange

- Initialize values, create fake (controllable) dependencies

Act

- Run the system under test, technically the method under test

Assert

- Check the return value or system status against expected criteria

Demo time :D

Never say: Unit tests, we do not have time for that!

Likewise, unit tests is not an excuse to not check your work!

Unit test tools

For **running** and **asserting**

- For back-end
  - MS Test, xUnit.net, Nunit, MbUnit, etc
- For front-end
  - Jasmine, Chai, Jest, Qunit

For **faking** dependencies

- For back-end
  - Moq, NSubstitue, MS Fakes, TypeMock, etc
- For front-end
  - Jasmine, SinonJS, etc

Final points

For production code, unit testing is not a choice, it is a **must**!

Writing testable code is more important that frameworks and tools for testing

Maybe using code coverage as a metric

Code- and Test-review to keep quality high

Bonus: TDD

TDD → Test Driven Development

It is a software development process

It is an approach to **design** and develop **testable** code

It is based around short 5-step cycles

1. Start by writing tests based on specifications and failure conditions
2. Run unit tests and verify the fail
3. Implement the actual methods
4. Run tests again and confirm they pass
5. Refactor code (I.e. clean up code, remove duplicates etc.)

Exercise time

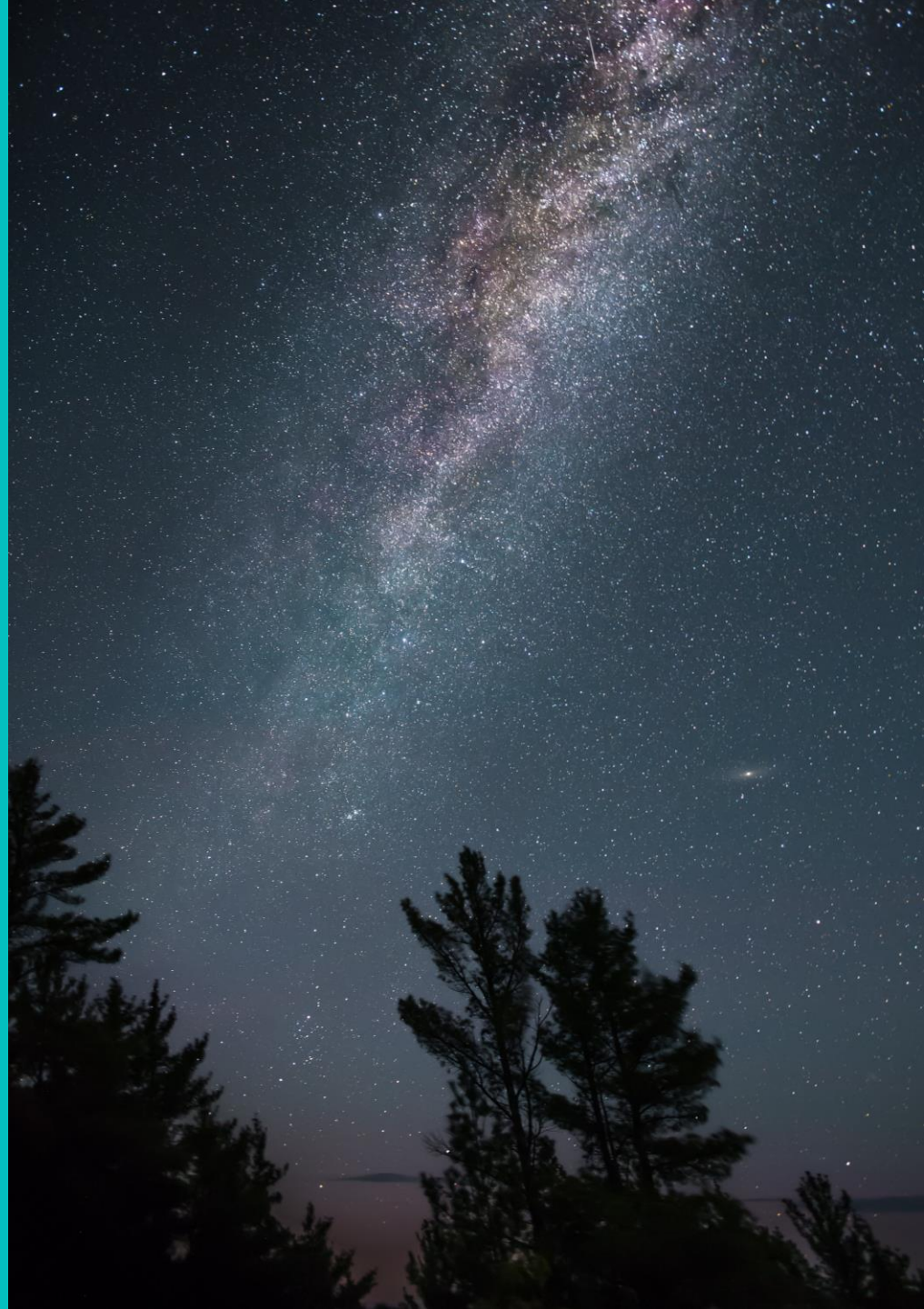
Grab the project from Code Examples called **TestingExercise**

Examine the project

Write unit tests for the home controller (A general example + controller test is provided)

Make sure to take into account the different ways thing can go wrong etc.

Security



Common web security vulnerabilities

Cross Site Scripting (XSS)

SQL Injection

Bad practices

Not using HTTPS

Broken authentication

Etc…

Reasons for vulnerabilities

Bad coding practices

Failure to properly validate input

Failure to sanitize input

How to handle user input

A talented attacker is able to change parts of our HTTP request.

I.e.

- Cookies
- Form Fields
- Hidden fields
- URL
- Headers

Therefore validation/sanitation of input should happen on both Client and Server

Cross site Scripting (XSS)

A security exploit where the attacker inserts malicious client-side code into a webpage

It has been around since the 1990s

Most major websites have been affected by XSS at some point

- Including Google, Yahoo and Facebook

Cross site Scripting (XSS) (cont.)

Preventing XSS attacks via:

- Filtering
- Input validation and output sanitization
- Use a safer browser
- Pay more attention when using shortened URLs
- Content Security Policies
  - Prevent this ☺

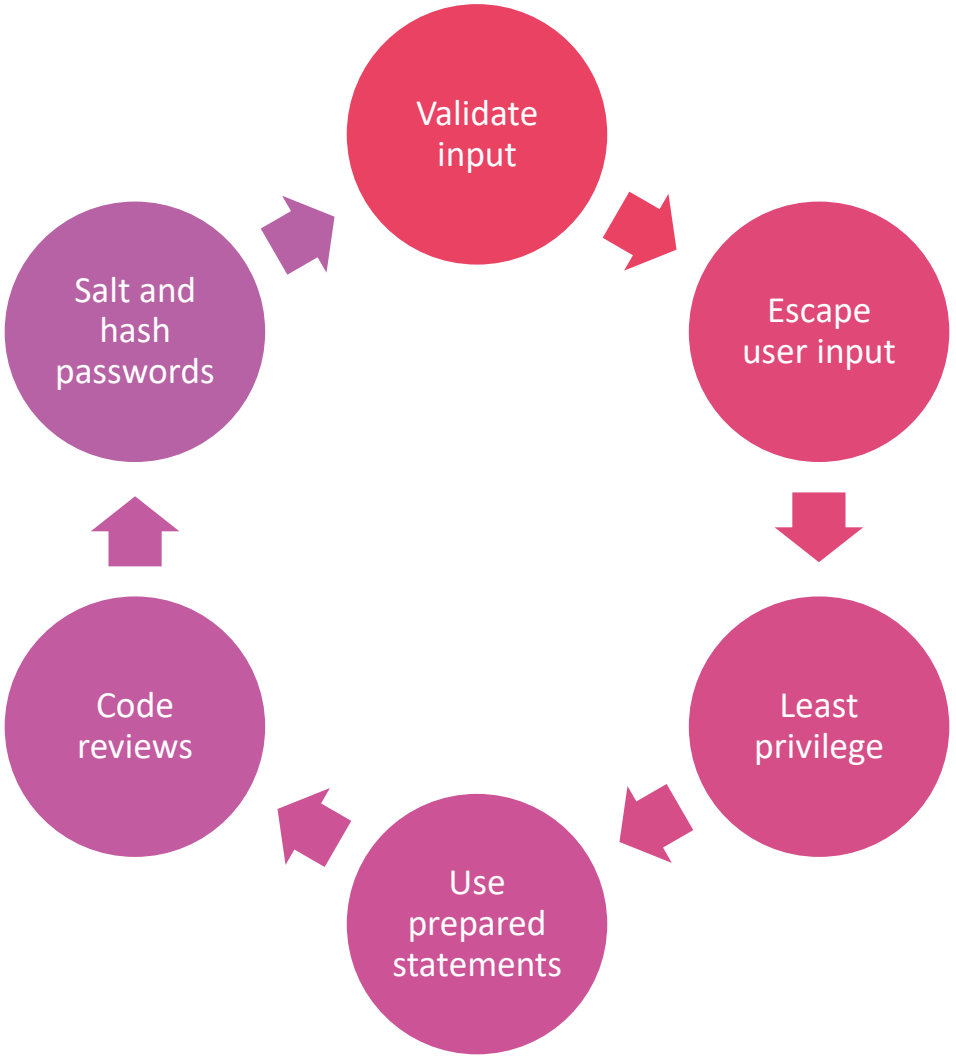
SQL Injection

SQL Injection is a technique where malicious users can inject SQL commands into an SQL statements via:

- URLs
- Input parameters
- Cookies
- Headers
- Etc.

SQL injection is a very old approach but is still popular

Mitigating risks



Questions?