Let's Tackle Software Testing

Motivation, Basics, Hands-on

symflower
AUTOMATING QUALITY ASSURANCE

Evelyn Haslinger eh@symflower.com
Markus Zimmermann mz@symflower.com

Agenda

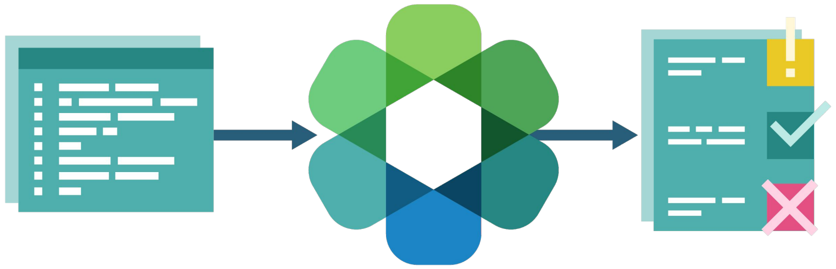
- **Introduction**
- The Problems of Software Testing
- Hands-On Examples
  - Property based testing (Fuzzing++)
  - Mutation testing
  - Model-based Testing
- Discussion (... for bigger questions)

Who Are We?

- Evelyn Haslinger
  - Research assistant @JKU
  - Senior software developer and scrum master @Sophos
- Markus Zimmermann
  - Writing software since primary school
  - Lots of “enterprise” applications, web services, tooling, software infrastructure, distributed apps and clustering, software testing
- **Now: Software testing and verification @Symflower**

What is Symflower?

Symflower completely **automatically writes**, runs and analyses **unit tests** revealing bugs, security issues and performance problems.



→ Reduce development and maintenance time

→ Increase quality of your software and tests

Who are you?

- Who are you?
- What is your experience with software testing?
- **What do you want to achieve today?**

Material for the workshop (You can also just watch and talk!)

- Repository
  https://github.com/symflower/sessions/2019/socrates-linz
- You need Docker (for executing examples)
  - Ubuntu, SUSE, https://docs.docker.com/install/
- Editor for editing Go (for editing examples)
  - (You can also just copy the code we prepared.)
  - https://github.com/golang/go/wiki/IDEsAndTextEditorPlugins
- Pull the Docker image (or build it yourself)
  - # docker pull symflower/socrates-linz-2019:latest
  - (or use one of our USB sticks)

Agenda

- Introduction
- **The Problems of Software Testing**
- Hands-On Examples
  - Property based testing (Fuzzing++)
  - Mutation testing
  - Model-based Testing
- Discussion (... for bigger questions)

1

The Problems of Software Testing

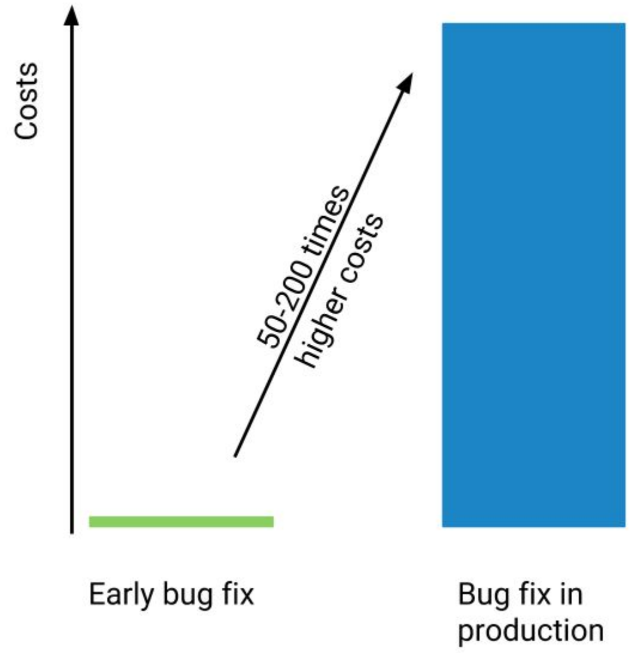
A quick overview

Some Fantastic Bugs(/Problems)

- Ariane 5 Flight 501 ($370.0M)
  - Reuse of code led to overflow 64bit->16bit
- Mars Climate Orbiter ($327.6M)
  - Expected different unit for metrics in one component
- Heartbleed Bug (>$500.0M)
  - Wrong bound-checking in kind of unused feature (now removed)
- Year 2000 problem (>$300.0B <- B as in billion!)
  - Huge part: software testing

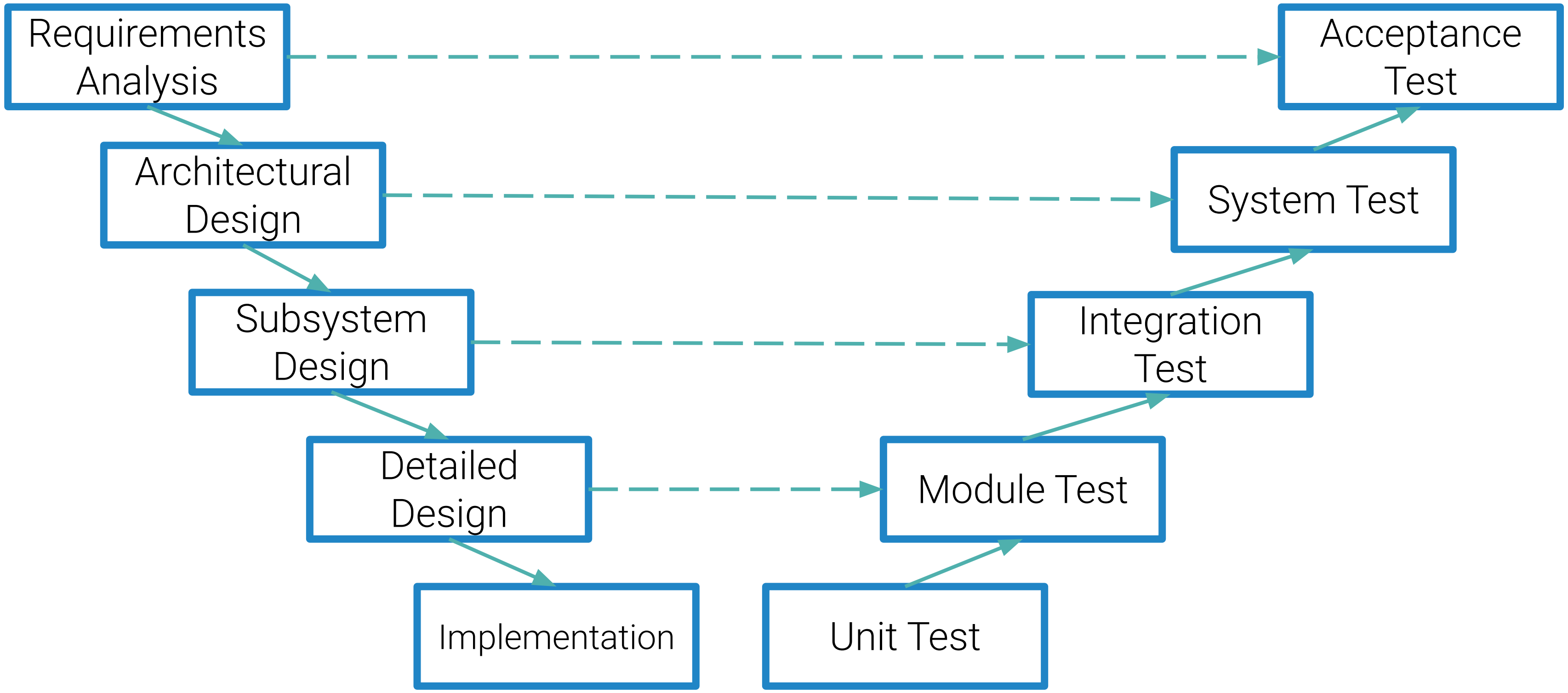
The Problem

- Software has errors
- Testing is time-consuming
- Humans are inaccurate



Over 30% of software development time is consumed by quality assurance.

Traditional Software Testing



Modern Software Testing

	Testing in Production		
Pre-Production	Deploy	Release	Post-Release
- Unit Tests	- Canarying	- Integration Tests	- Teeing
- Component Tests	- Dark Canaries	- Tap Compare	- Profiling
- Static Analysis	- Monitoring	- Load Tests	- Logs
- Coverage Tests	- Feature Flagging	- Shadowing	- Chaos Testing
- Benchmark Tests	- Exception Tracking	- Config Tests	- Monitoring
- Contract Tests	- Feature Graduation		- A/B Testing
- Acceptance Tests			- Tracing
- Smoke Tests			- Auditing
- UI/UX Tests			- OnCall Experience
- Penetration Tests			- Journey Tests

Spotted on https://twitter.com/samnewman/status/1176817869558034433

Goals

- 1) Early detection
  - Find problems before they go into production
  - Find problems **before they even go into staging**
- 2) Automate the complete testing process
  - Machines scale, humans do not
- 3) (Very very) thorough testing
  - Only feasible on the unit level (?)

Question: How are you reaching these goals?

~Implementation of 1&2

- CI/CD (Continuous Integration/Deployment)
  - Build, static analysis, automated tests, … per change
  - Automated deployments do not make mistakes
- “Testing” deployments
  - Find component/system problems on live environment
  - Copy “real” data to find outliers
  - Migrate deployment first to get ~real scenario
- CODE REVIEWS!

Problems with 2&3

- Am I testing the “right” things?
  - Specification-based testing?

(Java Code)

```java
static int compare(int a, int b) {
    int c = a - b;

    if (c < 0) {
        return -1;
    } else if (c > 0) {
        return 1;
    } else {
        return 0;
    }
}
```

(Java Code)

```java
static int compare(int a, int b) {
    int c = a - b;

    if (c < 0) {
        return -1;
    } else if (c > 0) {
        return 1;
    } else {
        return 0;
    }
}
```

Overflow, e.g. with a=0, b=-2147483648 -> c=-2147483648

Problems with 2&3

- Am I testing the “right” things?
    - Specification-based testing?
      (not enough, e.g. implementation diverges)

Problems with 2&3

- Am I testing the “right” things?
  - Specification-based testing?
    (not enough, e.g. implementation diverges)
- What is “thorough testing” anyway?
  - Should we look at code coverage?

Remember: Types of Code Coverage

- Statement coverage (usually line==statement coverage)
  - Each statement is executed
- Branch coverage
  - Each branch is once taken and once not taken
- Condition coverage
  - Each boolean subexpression evaluates once to true/false

Remember: Types of Code Coverage

- Modified condition/decision coverage (MC/DC)
  - Branch coverage
  - Condition coverage
  - Each condition affects decisions

```
if (a || b) && c {
    return 1;
}
```

```
a=false, b=true, c=false -> false
a=false, b=true, c=true  -> true
a=false, b=false, c=?    -> false
a=true, b=false, c=true  -> true
```

Is MC/DC Coverage Enough?

```
public static int division(int x, int y) {
    return x / y;
}
```

(Java Code)

Is MC/DC Coverage Enough?

```
public static int division(int x, int y) {
    return x / y;
}
```

$\leftarrow$ **y=0 leads to “division by zero”**

Is MC/DC Coverage Enough?

```java
public static int division(int x, int y) {
    return x / y;
}
```

y=0 leads to “division by zero”

… and there is an overflow:

x=-2147483648, y=-1 -> -2147483648

Clearly even MC/DC Coverage is not enough!

Problems with 2&3

- Am I testing the “right” things?
  - Specification-based testing?
    (not enough, e.g. implementation diverges)
- What is “thorough testing” anyway?
  - Is MC/DC coverage enough?
    (not enough, e.g. runtime errors)

Problems with 2&3

- Am I testing the "right" things?
  - Specification-based testing?
    (not enough, e.g. implementation diverges)
- What is "thorough testing" anyway?
  - Is MC/DC coverage enough?
    (not enough, e.g. runtime errors)
  - So, when do I know that I have tested enough?

Problems with 2&3

- Am I testing the “right” things?
  - Specification-based testing?
    (not enough, e.g. implementation diverges)
- What is “thorough testing” anyway?
  - Is MC/DC coverage enough?
    (not enough, e.g. runtime errors)
  - So, when do I know that I have tested enough?
- Clearly, manual creation of test cases it not enough
  - Can we automate the creation of test cases?

Suggestions to Approximate 2&3

- “Mutation Testing” for existing tests
  - Check if the whole implementation is **really** covered
  - E.g. https://github.com/zimmski/go-mutesting
- Mold implementation into test cases
  - One test case for every “interesting” path
  - Specification can then be checked with all cases
- Find (/generate) test cases
  - E.g. Fuzzing+MBT or better: Symflower

Agenda

- Introduction
- The Problems of Software Testing
- **Hands-On Examples**
  - Property based testing (Fuzzing++)
  - Mutation testing
  - Model-based Testing
- Discussion (... for bigger questions)

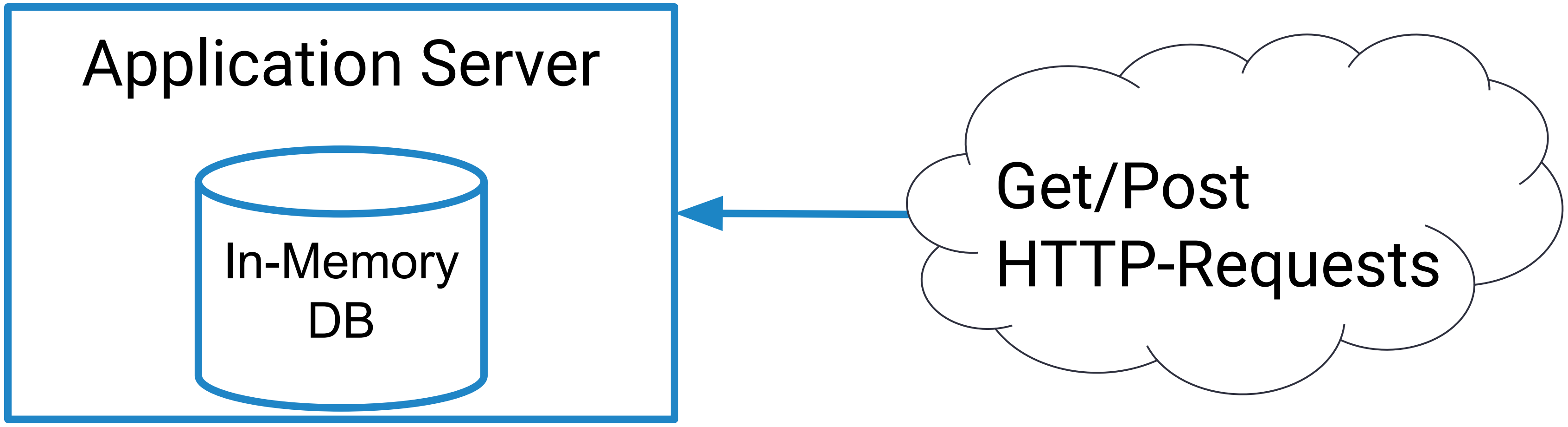
2

Hands-On Examples

Specification and high level overview

General Information

- Do not use code like this in production
- Application includes deliberate bugs
  - It includes severe security issues



Specification (1/4)

- The application offers the following features
  - User can register with a mail address and password
  - Registered user can post comments
  - Comments can be publicly viewed

Specification (2/4)

- Model User
  - Fields: mail address, password
  - Check if a user exists via mail
- Model Comment
  - Fields: mail address, created, message
  - Created timestamp is generated

Specification (3/4)

- Route GET /
  - Shows all comments and a form to insert a new comment
- Route POST /
  - Creates a new comment
- ROUTE GET /register
  - Shows a form to register a user
- ROUTE POST/register
  - Creates a new user

Specification (3/4)

- Our application is massively used…
  - … but also misused
- We want to filter “swear words
- This change have us worried
  - How can we make sure that we do not break X?
  - How can we make sure that new Y works?

We are thinking about better ways of testing

What and How Should We Test?

- What kinds of tests would you do?
- What test cases/scenarios would you do?

Agenda

- Introduction
- The Problems of Software Testing
- Hands-On Examples
  - **Property based testing (Fuzzing++)**
  - **Mutation testing**
  - **Model-based Testing**
- Discussion (... for bigger questions)

Agenda

- Introduction
- The Problems of Software Testing
- Hands-On Examples
  - Property based testing (Fuzzing++)
  - Mutation testing
  - Model-based Testing
- **Discussion (... for bigger questions)**

6
Discussion

symflower
AUTOMATING QUALITY ASSURANCE

Evelyn Haslinger eh@symflower.com
Markus Zimmermann mz@symflower.com

We Are Hiring!

- We offer
  - challenging algorithmic tasks to work on
  - state of the art development processes and tools
  - a work environment that you can shape with us
- Talk to us after the lecture
- Drop us an email with your CV at you@symflower.com
- Please recommend us to your peers

symflower

AUTOMATING QUALITY ASSURANCE

Evelyn Haslinger eh@symflower.com
Markus Zimmermann mz@symflower.com