Finding Regressions in Mission Critical Software Workflows

San Francisco Bay Area C++ User Group, January 10, 2023

Pejman Ghorbanzade

pejman@touca.io

This talk is opinionated.

About Me

- Founder and CEO of Touca.io (Techstars '22)
- Canon Medical Informatics
- VMware Carbon Black
- Digi International
- Passionate about maintaining software at scale

Software Engineering

- Programming
  - Theoretical problem solving
  - Like sport
- Software Engineering
  - Problem solving within business constraints
  - Like gardening

> "Software engineering is programming integrated over time." - Titus Winters

The Building that Moved

Software Gardening

- Why building software is just like building a house
- Why software development is not like building a house

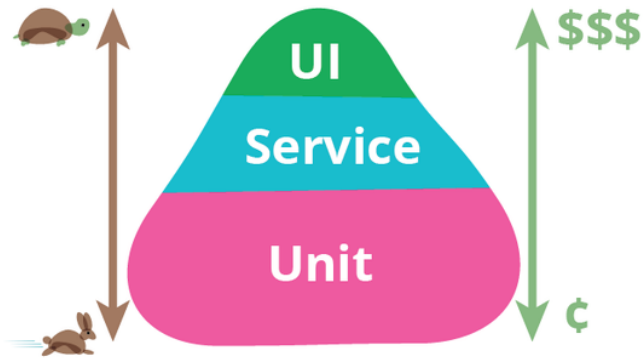
> Software is a tractable medium.

> "The most helpful thing I learnt from chess is to make good decisions on incomplete data in a limited amount of time." - Magnus Carlsen


Gems - Museum
Share
Watch on YouTube

Software Testing

- Good tests are:
  - Cheap to Write
  - Easy to Read
  - Fast to Run
  - Easy to Change



> Good tests have high return on investment.

> Good tests give you confidence when they pass and teach you something when they fail.

Question

How do you measure the return on investment of software testing?

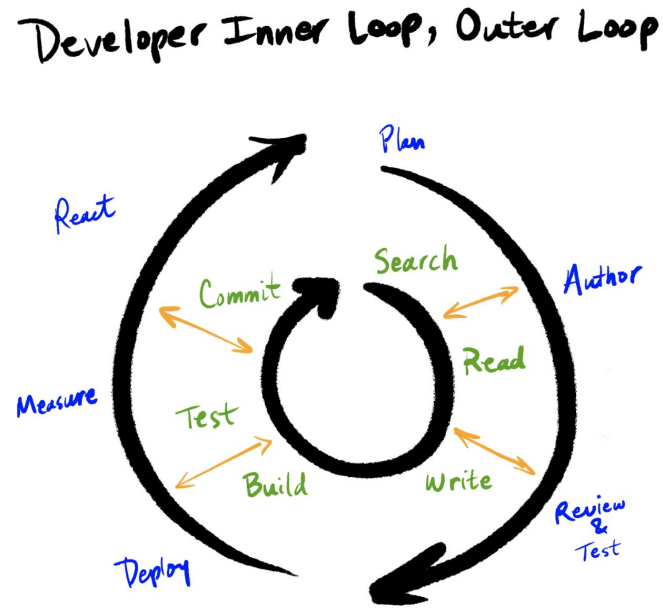
The Cost of a Bug

Finding bugs after deployment	💰💰💰💰💰
Finding bugs before release	💰💰💰
Finding bugs during QA testing	💰💰
Finding bugs during code review	💰
Finding bugs during development	

Developer Productivity

- Depth: Does this test provide adequate confidence?
- Speed: Does this test provide timely feedback?

> Confidence level and feedback cycle are non-const variables.



Fun Fact

Most engineering teams think that they are doing worse than average in following industry best practices.

Hot Take

Improving software testing is a _____ problem.

1. business
2. culture
3. design
4. tooling

Improving Culture

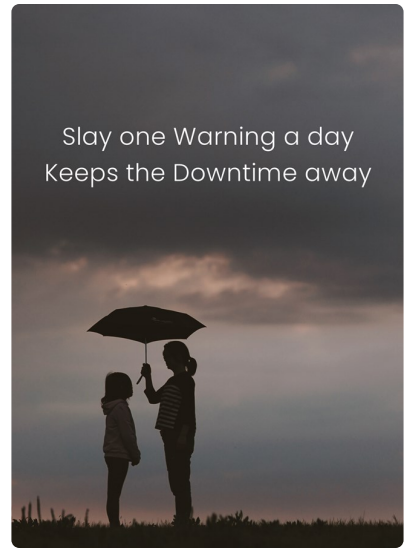
Active Debate

- Regular technical debt review
- Periodic code review policy
- Pair/Ensemble programming sessions
- Regular knowledge hand-off sessions
- Collecting and sharing software quality metrics

Improving Culture

No Broken Windows

In criminology, the broken windows theory states that visible signs of crime, anti-social behavior and civil disorder create an urban environment that encourages further crime and disorder, including serious crimes. The theory suggests that policing methods that target minor crimes such as vandalism, loitering, public drinking, jaywalking, and fare evasion help to create an atmosphere of order and lawfulness.



Fun Fact

It takes 23 days for software engineers to gain confidence that a given code change works as they expect.

The Problem

How can we refactor half a million lines of code without causing any side effects?

Candidate Solution A

```
auto new_output = new_system(testcase);
auto old_output = old_system(testcase);
compare(new_output, old_output);
```

Disadvantages

- Test is difficult to setup
- Test system is inefficient to run
- Test system is not reusable

Candidate Solution B

```
auto new_output = new_system(testcase);
auto new_file = write_to_file(testcase, new_output);
auto old_file = find_old_file(testcase);
compare(new_file, new_output);
```

Disadvantages

- Dealing with files is no fun
- Test system is hard to maintain
- Test system is not reusable

Demo Time

Approval Testing

pejman@touca.io

Candidate Solution C

```
auto new_output = new_system(testcase);
auto new_description = describe(new_output);
submit(testcase, new_description);
```

Disadvantages

- Limited customization
- Overkill for small projects
- Requires remote computing resources

Example

Code Under Test

```
struct Student {
  std::string username;
  std::string fullname;
  Date birth_date;
  std::vector<Course> courses;
};

Student find_student(const std::string& username);
```

Example

Regression Test

```
#include "students.hpp"
#include "touca/touca.hpp"

int main(int argc, char* argv[]) {
  touca::workflow("students", [](const std::string& username) {
    const auto& student = find_student(username);
    touca::check("username", student.username);
    touca::check("fullname", student.fullname);
    touca::check("birth_date", student.birth_date);
    touca::check("courses", student.courses);
  });
  touca::run(argc, argv);
}
```

Example

Serializing Custom Types

```
template <>
struct touca::serializer<Date> {
  data_point serialize(const Date& value) {
    return object("Date")
        .add("year", value.year)
        .add("month", value.month)
        .add("day", value.day);
  }
};
```

👉 CppCon2021: "Building an Extensible Type Serialization System Using Partial Template Specialization", Pejman Ghorbanzade

Design Requirements

- Intuitive developer experience
- Intrinsic support for common types
  - Must support integral types, fractional types, Strings, Iterables, and other common standard types
- Extensible design to support user-defined types
  - Must allow users to introduce logic for handling custom types

Demo Time

pejman@touca.io

Questions

- https://touca.io
- https://github.com/trytouca/trytouca
- https://fosstodon.org/@heypejman
- https://linkedin.com/in/ghorbanzade
- pejman@touca.io