Software testing: new insights & next steps

Andy Zaidman


Software Failures

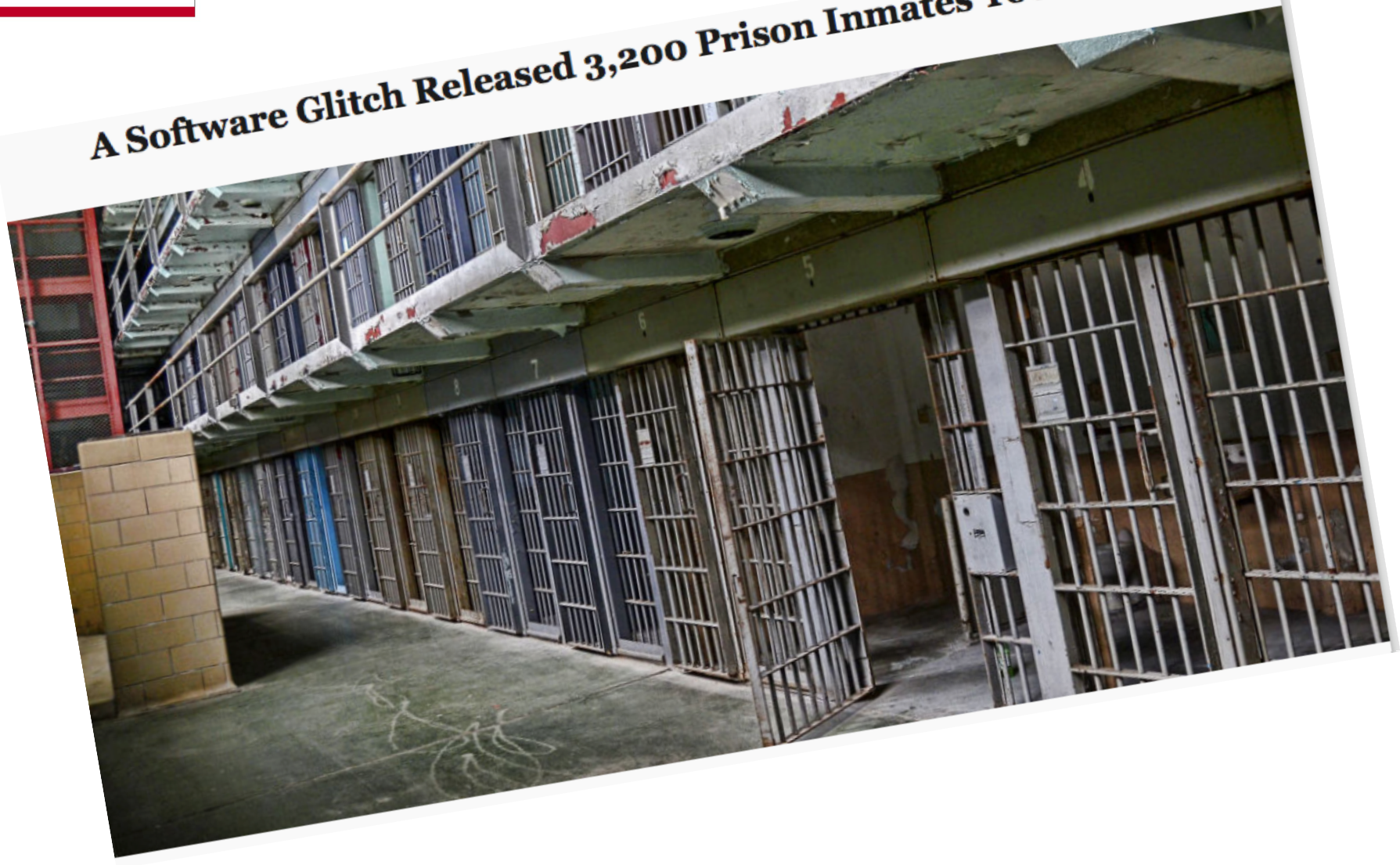
Many software failures each year…

A 2002 estimate: software failures cost the US economy $59.5 billion

Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing. Technical report, National Institute of Standards and Technology RTI Project.

Some key examples from december 2015…

A Software Glitch Released 3,200 Prison Inmates Too Early



http://www.popularmechanics.com/technology/infrastructure/a18721/washington-state-released-3200-inmates-early/

Good News: Computer Glitch Caused The IRS To Issue $46 Million Worth Of Questionable Refunds

"Sjoemelsoftware" woord van het jaar in Nederland

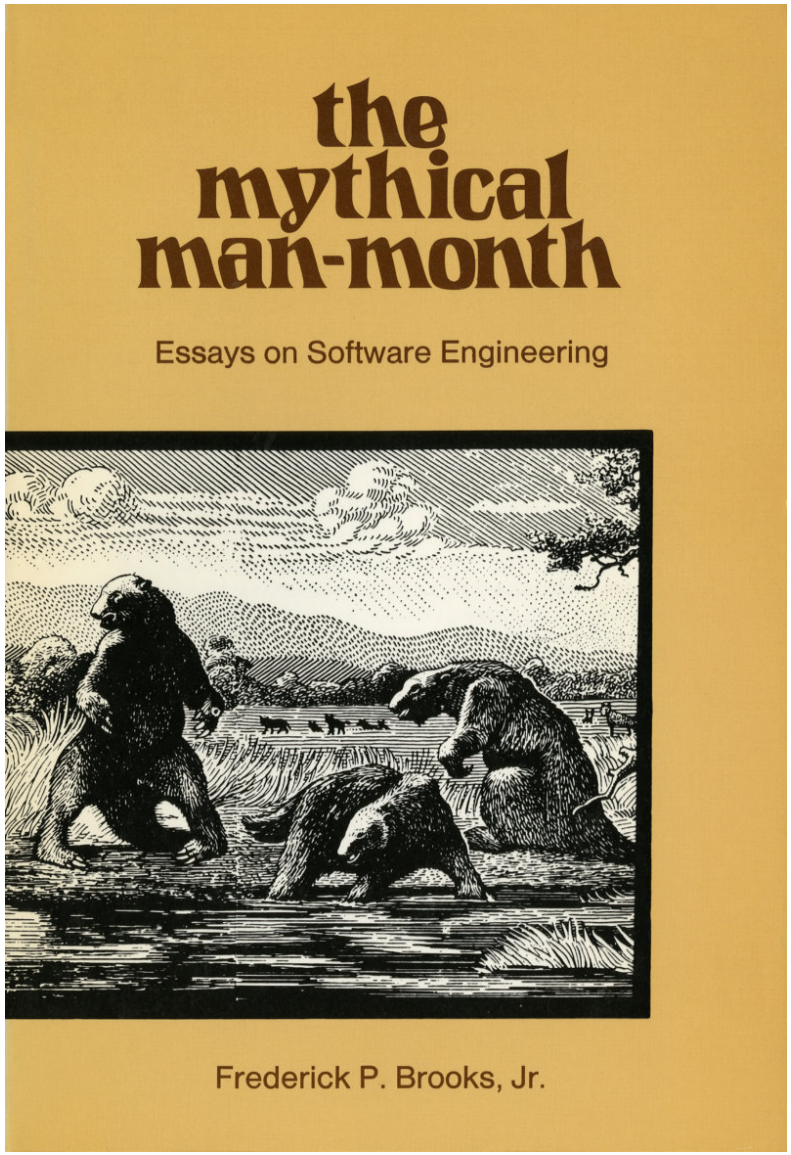
As software engineers…

What do we know?

What can we do?

Let’s start with the “knows”

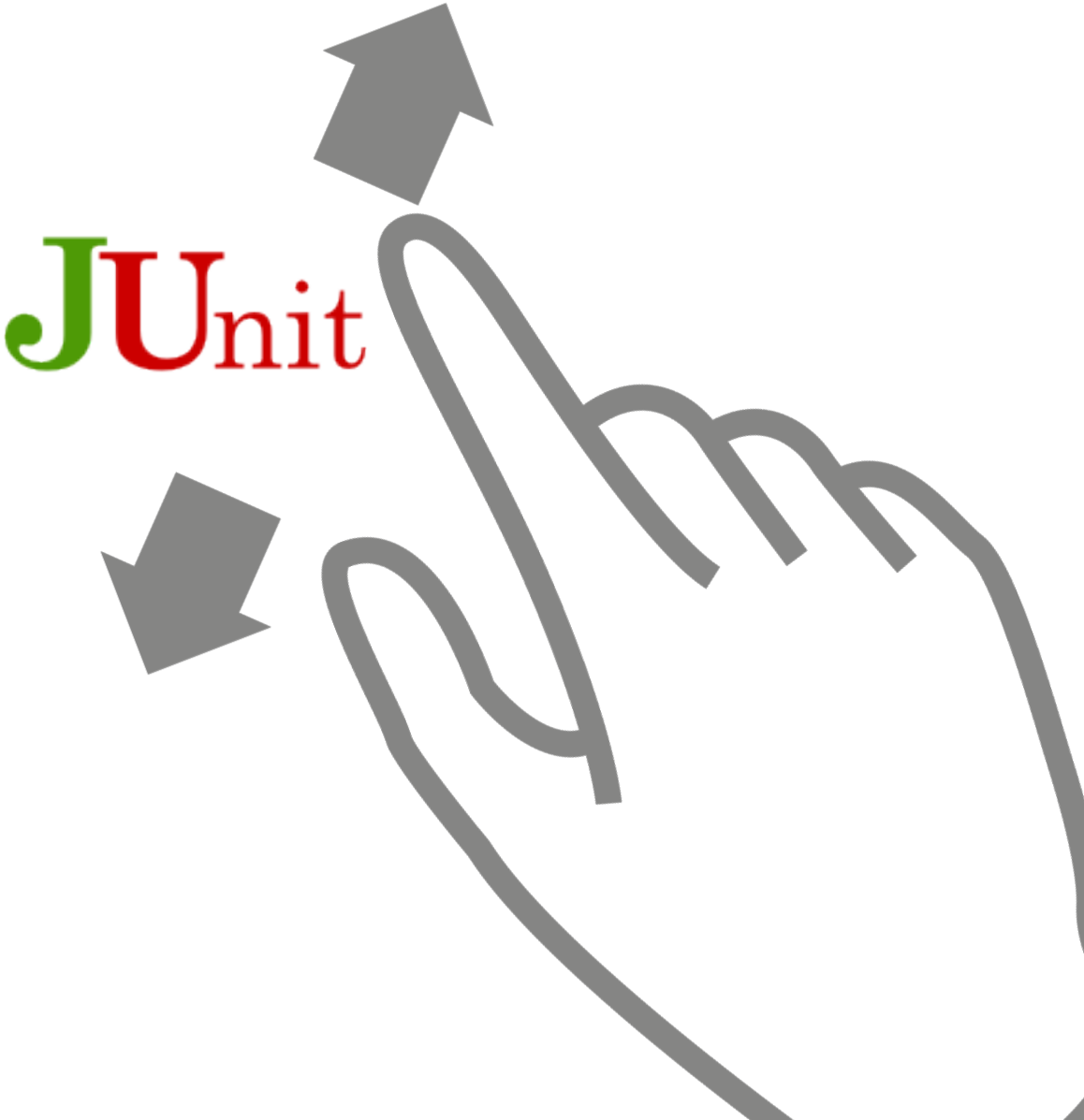
The Mythical Man-Month



Brooks “observed” that **testing and debugging take 50% of the time**

How?
“Calendar research”

When
± 40 years ago…


JUnit

Research Tool	Software Analytics
"Big Brother" in your IDE → measures testing activities • 416 Softw. Engineers, 68 countries • Java / Eclipse • 13.7 man years of development observed	Immediate feedback for developers

If we ask 416 SW engineers…

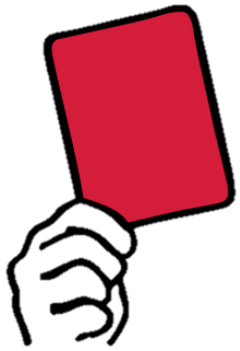
Can you estimate how much time you spend on engineering **test code** versus **production code?**

50% test code

50% production code

30% test code

70% production code



WatchDog Pre-Test Questionnaire

Can you estimate how much time you spend on engineering **test code** versus **production code?**

48% - 52%

test code production code

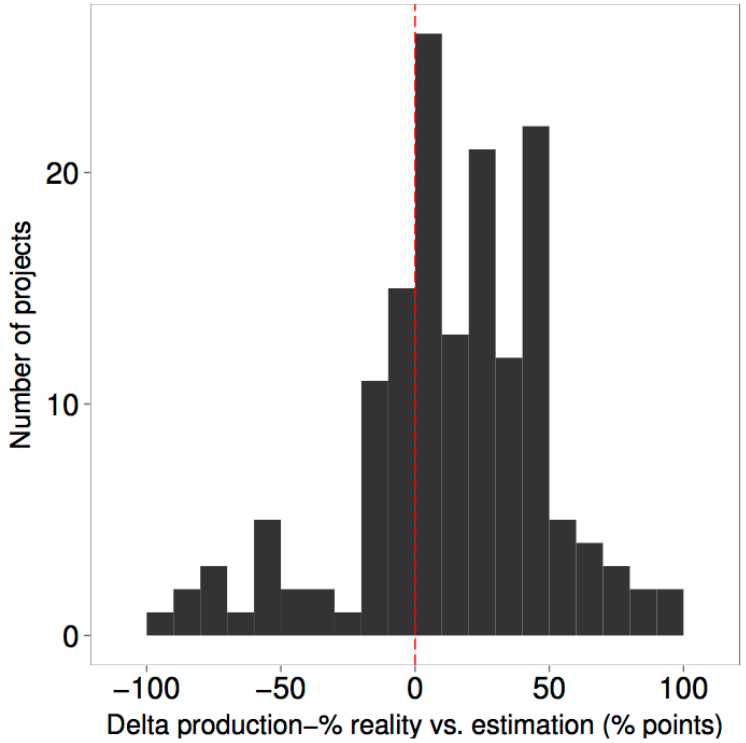
After measuring ≥ 3 months

25% - 75%

test code engineering

production code engineering

Most participants overestimate their test engineering activities



So, we should test more?

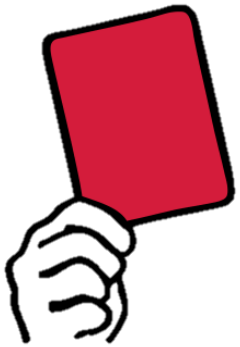
No test engineering

For projects with test code:
53% of developer did not execute, read, or modify a single test within five months.

Do sets of unit test executions take...

< 5 seconds

≥ 5 seconds

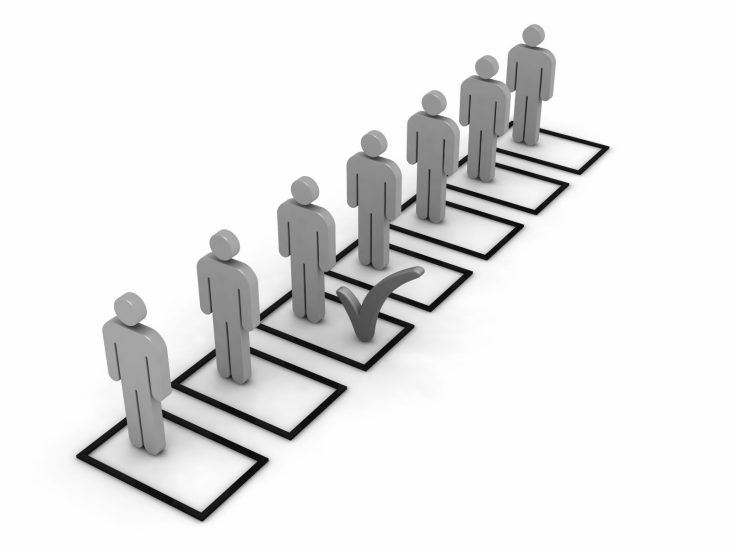


Swift tests

Execution of set of unit tests takes 0.54 seconds (median).

Ad hoc test selection

Developers frequently select a specific set of tests to run in the IDE. In most cases, developers execute one single test.

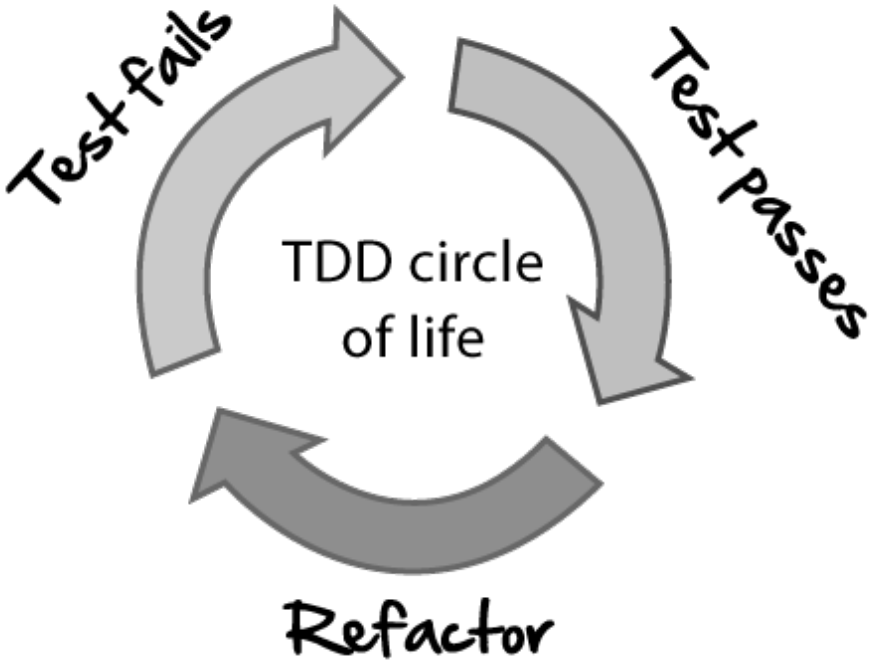


Tests fail

65% of test executions in the IDE fail.

Test Driven Development

TDD is not widely practiced. Programmers who claim to do it, neither follow it strictly nor for all their modifications.



What do we do when a test fails?

- In 13% of the cases we simply delete the test without looking any further…
  → easy way out

Are we really set on **breaking** the software?

→ SW engineers tend to check if something is true, not whether something is false…

Gül Çalıklı, Ayse Bener: Influence of confirmation biases of developers on software quality: an empirical study

Do testing and continuous integration really go hand-in-hand?

- > 50% of build failures are due to testing failures
- CI can really be useful
  - You can build/run in different environments (different OS, different version of Java, …)
  - In 10% of multi-environment integrations, at least one integration fails
- CI = last line of defense?
  (→ see later in the slides as well)

How can we (re)act?

1. Modern Code Reviews

- Modern Code Reviews (MCRs) are lightweight code inspections aimed at finding issues in code
- Are said to be complementary issues w.r.t. issues found through automated testing

- Why ”Modern”? Code reviews have been around since long and originate from book by Fagan. Fagan style means code reviewing takes on a very detailed (and expensive) form

Are code reviews that good?

- Recently MCRs have been shown to be especially good at finding maintainability defects ←→ functional defects
  - > 80% of defects found during MCR are maintenance related

2. Don’t blindly trust code coverage

- Code coverage metrics can be deceiving
  - I can write one high-level test to reach 80% coverage
  - I can write 100 unit tests to reach 80% coverage

→which situation is better?

- Does code coverage say something about e.g. boundary testing? Or negative testing?

So maybe mutation analysis?

- Maybe mutation analysis and test coverage should be used hand-in-hand?

- Should we then still bother with coverage information?
  - What would happen if we apply mutations with 0% branch coverage?

3. Don’t fool yourself with code coverage

- Measure code coverage separately at several levels of granularity
  - Unit testing / Integration testing
  - System testing

- These tests have a different purpose, treat them separately!

4. Is CI the last line of defense?

- Why not take it further and go towards Continuous Deployment (CD)?
  - Even if it is only for testing…
- Try to recreate the environment as closely as possible, create Docker containers and start running tests in these containers

5. ”Shift right”

- In modern SOA systems, not everything can be tested before deployment
- Continuous monitoring of system
  → online testing

- Example: *Spectrum-Based Fault Localisation* is a technique to identify faulty services in a running service-based system
  - SBFL tries to find that particular service which is responsible for the root cause

Spectrum-Based Fault Localisation

Comp.	Character counter	t_1	t_2	t_3	t_4	t_5	t_6	SC_o
	public int count(String s){	[Activity Matrix]						
C_0	int upper = 0 ; int lower = 0; int digit = 0; int other = 0;	1	1	1	1	1	1	0.82
C_1	for(int i = 0; i¡s.length(); i++){	1	1	1	1	1	1	0.82
C_2	char c = s.charAt(i);	1	1	1	1	1	1	0.82
C_3	if(c >= 'A' && c <= 'Z')	1	1	1	1	0	1	0.89
C_4	**upper += 2;**	1	1	1	1	0	0	1.00
C_5	else if(c >= 'a' && c <= 'z')	1	1	1	1	0	1	0.89
C_6	lower++;	1	1	0	0	0	0	0.71
C_7	else if(Character.isDigit(c))	1	0	1	0	0	1	0.58
C_8	digit++; }	1	0	1	0	0	1	0.58
C_9	other = s.length()-upper-lower-digit;	1	1	1	1	1	1	0.82
C_{10}	return other;	1	1	1	1	1	1	0.82
	}							
	Output vector (verdicts)	1	1	1	1	0	0	

6. End-to-end tests

- Test right from the UI to the back-end of the application
- Automate with tools like Selenium, Robotium, Appium
  - Capture-replay
  - Record the user actions in the user interfaces, the specific input values, etc.
  - See what the next screen brings and whether that matches expectations

7. Does test code quality matter?

Who agrees with the following statements?

1. Test code does not have to be maintained

Data obtained from 75 years of development of 18 projects with 110000 defects and 49000 feature requests

7. Does test code quality matter?

Who agrees with the following statements?

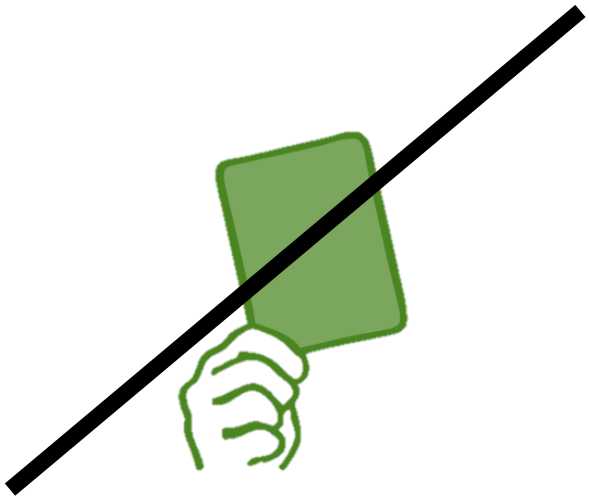
2. Nobody is reading test code, so I should not spend effort on making it “clean”

Consequence of test code quality

1. Having high quality test code allows me to solve bugs more quickly

Consequence of test code quality

1. Having high quality test code allows me to solve bugs more quickly



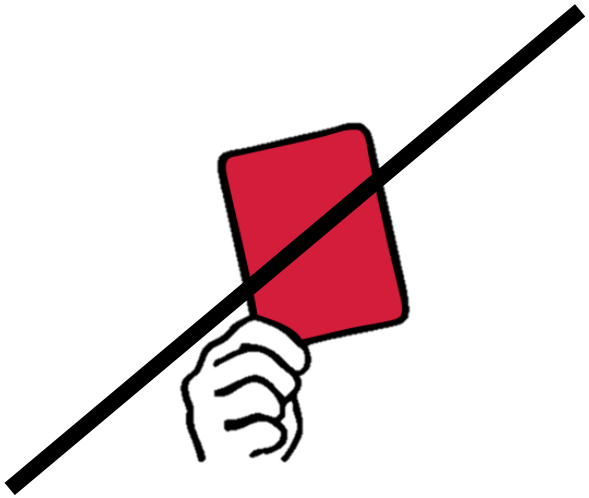
Why?

Consequence of test code quality

2. Having high quality test code allows me to implement new features more quickly

Consequence of test code quality

2. Having high quality test code allows me to implement new features more quickly



Why?

8. Behavior Driven Development

- Acceptance testing = test conducted to determine if the requirements are met

- What if the requirements are tests?
  - Can the client read these requirements?
  - Can these requirements be executed?

  → behavior driven development (BDD)

BDD

Given a context
When an event happens
Then an outcome should occur

Scenario 1:
Refunded items should be returned to stock

Given that a customer previously bought a black sweater from me
And I have three black sweaters in stock.
When he returns the black sweater for a refund
Then I should have four black sweaters in stock.

```
public class SalarySteps {

SalaryManager manager;

@Given("^the salary management system is initialized with the following data$")

public void the_salary_management_system_is_initialized_with_the_following_data(
        final List<Employee> employees) throws Throwable {
                manager = new SalaryManager(employees);
        }

@When("^the boss increases the salary for the employee with id '(\\d+)' by (\\d+)%$")

public void the_boss_increases_the_salary_for_the_employee_with_id_by(
        final int id, final int increaseInPercent) throws Throwable {
                manager.increaseSalary(id, increaseInPercent);
        }

@Then("^the payroll for the employee with id '(\\d+)' should display a salary of
(\\d+)$")

public void the_payroll_for_the_employee_with_id_should_display_a_salary_of(
        final int id, final float salary) throws Throwable {
                Employee nominee = manager.getPayroll(id);
                assertThat(nominee.getSalary(), equalTo(salary));
        }
}
```

Questions?