

Testing

Antonio Brogi

Department of Computer Science
University of Pisa



Introduction



https://www.youtube.com/watch?v=PK_yguLapgA [0,2:08]

«This was caused by an internal variable related to the horizontal velocity of the launcher exceeding a limit which existed in the software of this computer.»

Testing

- 1) Execute your program using data that simulate user inputs
- 2) Observe program's behaviour

Behaviour is what you expect → test succeeded

Behaviour is NOT what you expect → test failed

Types of testing

Functional testing



Test functionality of overall system to discover bugs

User testing

Test that product is useful to and usable by end users

Alpha testing: "Do you really want these features?"

Beta testing: Early version of product distributed to users to check product usability and effective interoperability

Performance and load testing

Test that system responds quickly to service requests

Test that system can handle different loads and scales gracefully as the load increases

Security testing



Find vulnerabilities that attackers may exploit

Warning (1/2)



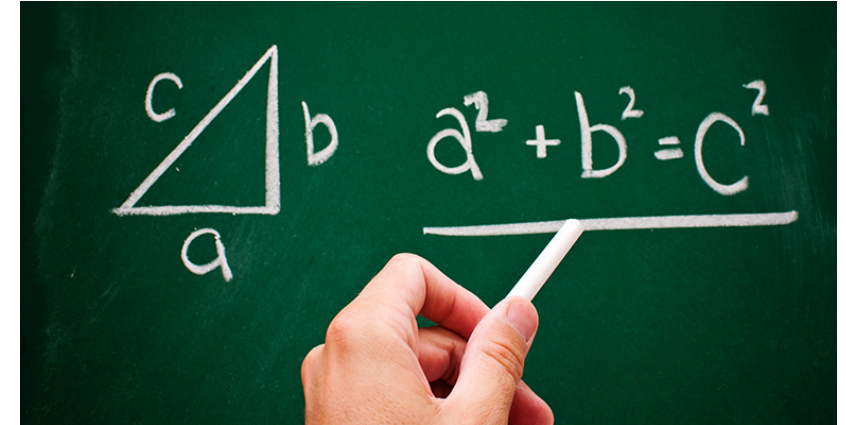
“Program testing can be used to show the presence of bugs, but never to show their absence!”

— Edsger W. Dijkstra

Warning (2/2)



≠



software testing ≠ software verification

Introduction

Functional testing

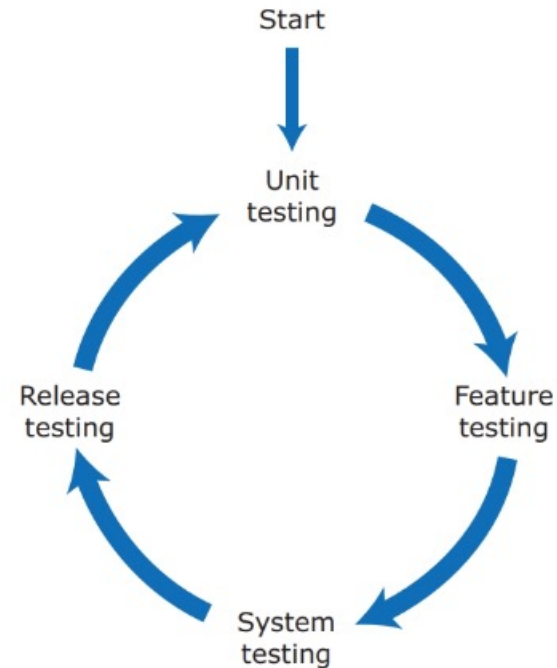
Functional testing

Large set of program tests so that all code is executed at least once

Testing should start on the day you start writing code

Develop/test cycle simplified by automated tests

Functional testing is a staged activity



Unit testing

Test program units (e.g. function, method) in isolation

Unit testing principle

If a program unit behaves as expected for a set of inputs that have some shared characteristics, it will behave in the same way for a larger set whose members share these characteristics.

Example

If your program behaves correctly on the input set {1, 5, 17, 45, 99}, you may conclude that it will also process all other integers in the range 1 to 99 correctly

Unit testing: Equivalence partitions

Identify “equivalence partitions”: sets of inputs that will be treated the same in your code

Test your program using several inputs from each equivalence partition

Programmers make mistakes at boundaries: Identify partition boundaries and choose inputs at these boundaries

Unit testing: Guidelines

Guideline	Explanation
Test edge cases	If your partition has upper and lower bounds (e.g., length of strings, numbers, etc.), choose inputs at the edges of the range.
Force errors	Choose test inputs that force the system to generate all error messages. Choose test inputs that should generate invalid outputs.
Fill buffers	Choose test inputs that cause all input buffers to overflow.
Repeat yourself	Repeat the same test input or series of inputs several times.
Overflow and underflow	If your program does numeric calculations, choose test inputs that cause it to calculate very large or very small numbers.
Don't forget null and zero	If your program uses pointers or strings, always test with null pointers and strings. If you use sequences, test with an empty sequence. For numeric inputs, always test with zero.
Keep count	When dealing with lists and list transformations, keep count of the number of elements in each list and check that these are consistent after each transformation.
One is different	If your program deals with sequences, always test with sequences that have a single value.



Feature testing

A product feature implements some useful user functionality

Features must be tested to show that functionality

- (1) is implemented as expected and
- (2) meets the real needs of users

Feature normally implemented by multiple interacting program units

Two types of tests

(1) Interaction tests

Testing interactions between units (developed by different developers)
Can also reveal bugs in program units that were not exposed by unit testing

(2) Usefulness tests

Testing that feature implements what users are likely to want
Product manager should be involved in designing usefulness tests

Feature testing

Story title	User story
User registration	As a user, I want to be able to log in without creating a new account so that I don't have to remember another login ID and password.
Information sharing	As a user, I want to know what information you will share with other companies. I want to be able to cancel my registration if I don't want to share this information.
Email choice	As a user, I want to be able to choose the types of email that I'll get from you when I register for an account.



From scenarios/user stories to feature tests

Test	Description
Initial login screen	Test that the screen displaying a request for Google account credentials is correctly displayed when a user clicks on the "Sign-in with Google" link. Test that the login is completed if the user is already logged in to Google.
Incorrect credentials	Test that the error message and retry screen are displayed if the user inputs incorrect Google credentials.
Shared information	Test that the information shared with Google is displayed, along with a cancel or confirm option. Test that the registration is canceled if the cancel option is chosen.
Email opt-in	Test that the user is offered a menu of options for email information and can choose multiple items to opt in to emails. Test that the user is not registered for any emails if no options are selected.

System testing

Testing the system as a whole

- To discover unexpected/unwanted interactions between the features
- To discover if system features work together effectively to support what users really want to do
- To make sure system operates as expected in the different environments where it will be used
- To test responsiveness, throughput, security, and other quality attributes

Tip: Use a set of scenarios/user stories to identify users' end-to-end pathways

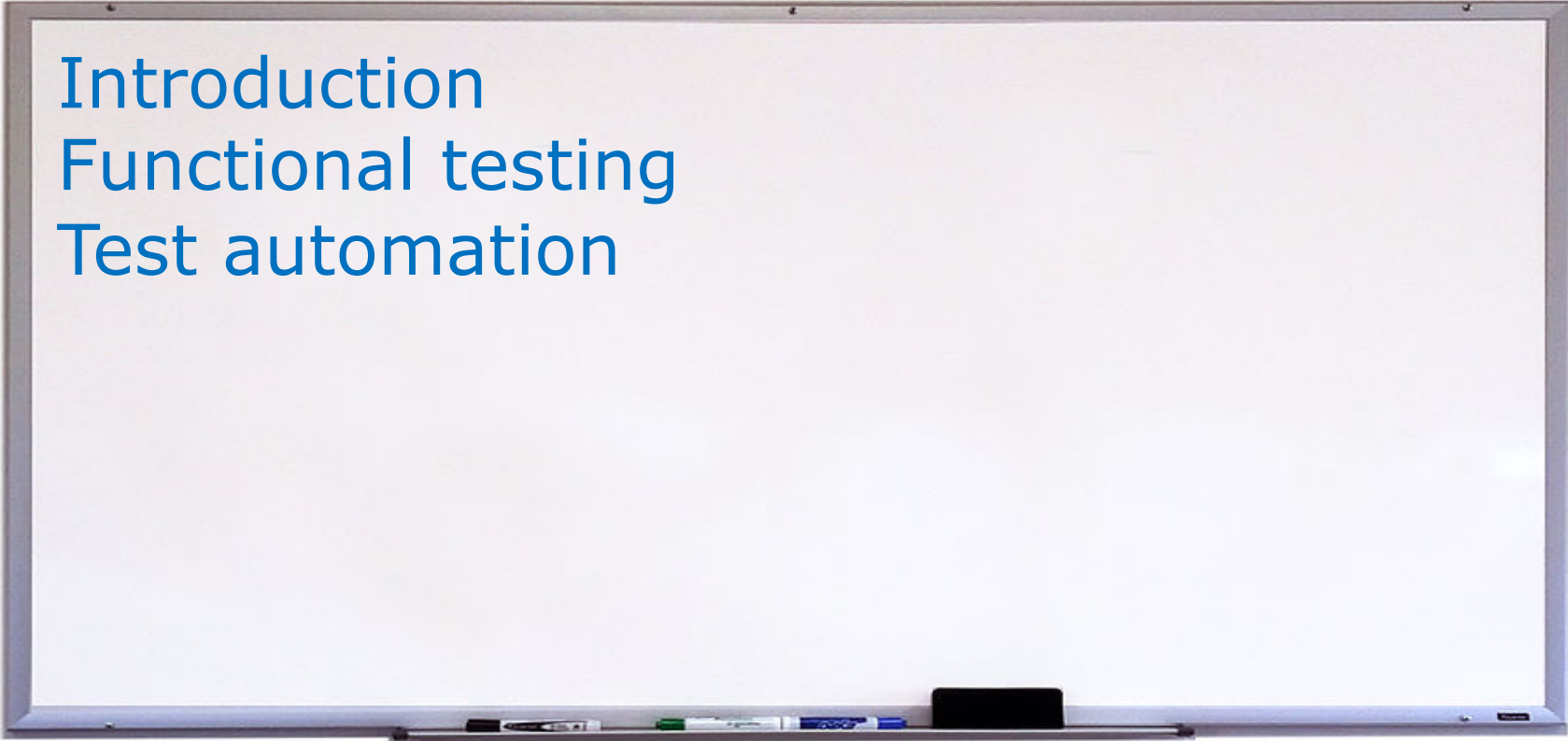
Release testing

Testing a system that is intended for release to customers

- Release testing tests the system in its real operational environment (rather than in a test environment)
- The aim of is to decide if the system is good enough to release, not to detect bugs in the system

Needed since preparing a system for release involves packaging the system for deployment, installing used software and libraries, configure parameters – and mistakes can be made in that process

If you deploy on the cloud, an automated continuous release process can be used



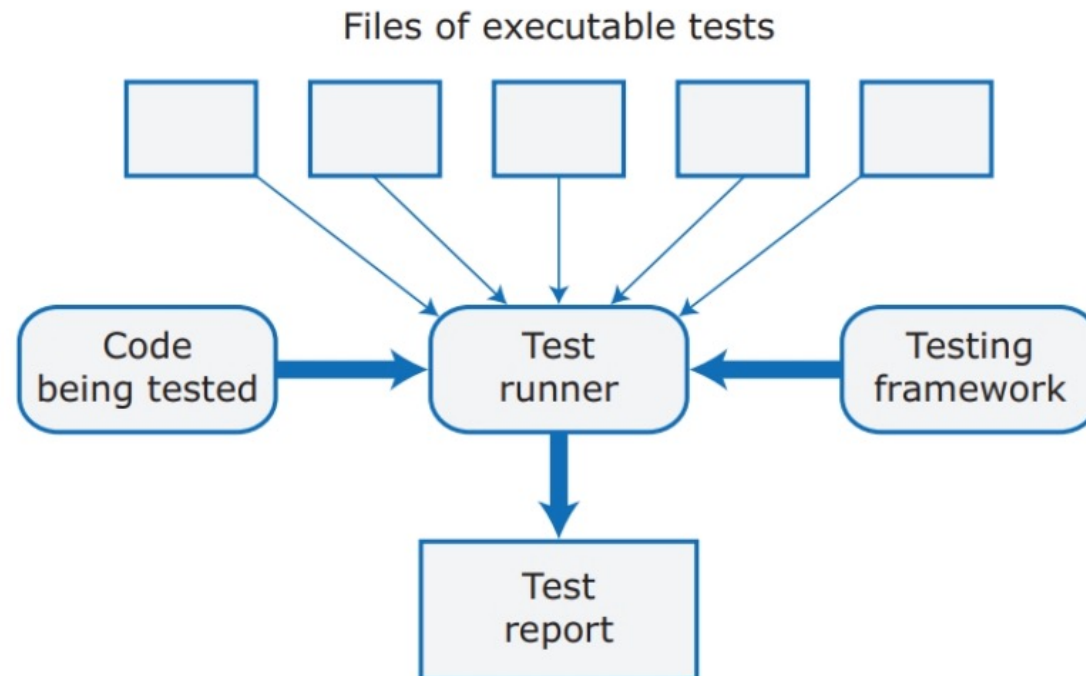
Introduction
Functional testing
Test automation

Test automation

Automated testing widely used in product development companies

Testing frameworks available for all widely used programming languages

Executable tests check that software return expected result for input data



Test automation

Good practice: Structure automated tests in three parts

1. Arrange - Set up the system to run the test (define test parameters, mock objects if needed)
2. Action - Call the unit that is being tested with the test parameters
3. Assert – Assert what should hold if test executed successfully.

```
#Arrange - set up the test parameters
```

```
p = 0
```

```
r = 3
```

```
n = 31
```

```
result_should_be = 0
```

```
#Action - Call the method to be tested
```

```
interest = interest_calculator (p, r, n)
```

```
#Assert - test what should be true
```

```
self.assertEqual (result_should_be, interest)
```

Test automation

Test code can include bugs!

Good practice to reduce the chances of test errors:

1. Make tests as simple as possible
2. Review all tests along with the code that they test

Test automation

Unit tests are the easiest to automate

Good unit tests reduce (do not eliminate) need of feature tests

GUI-based testing expensive to automate, API-based testing preferable

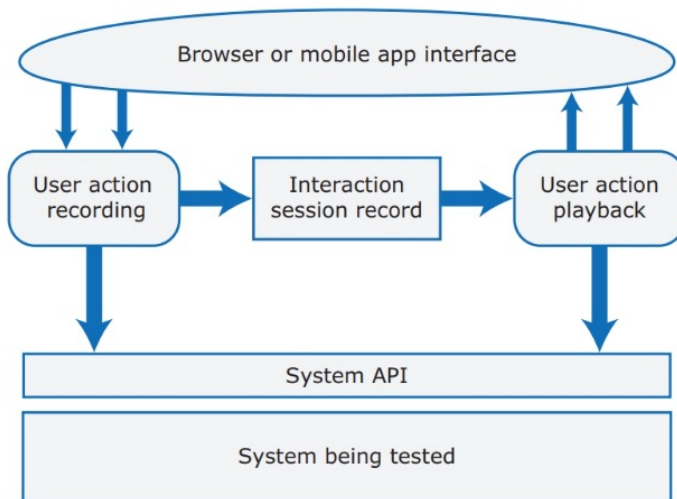
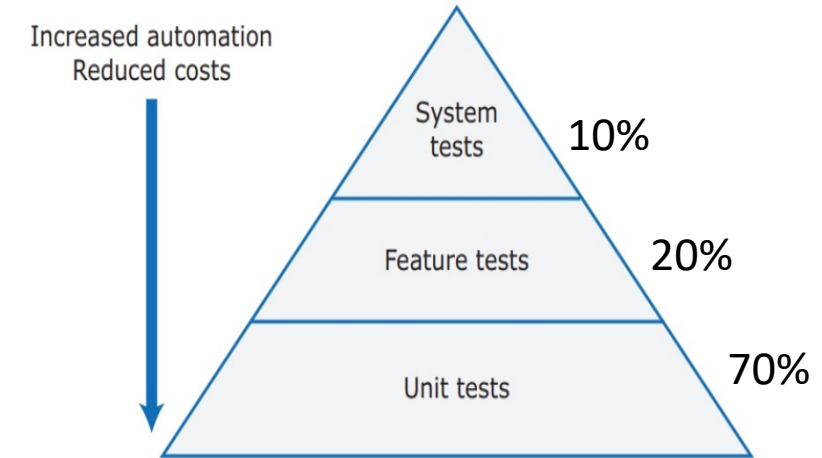
Need multiple assertions to check that feature executed as expected

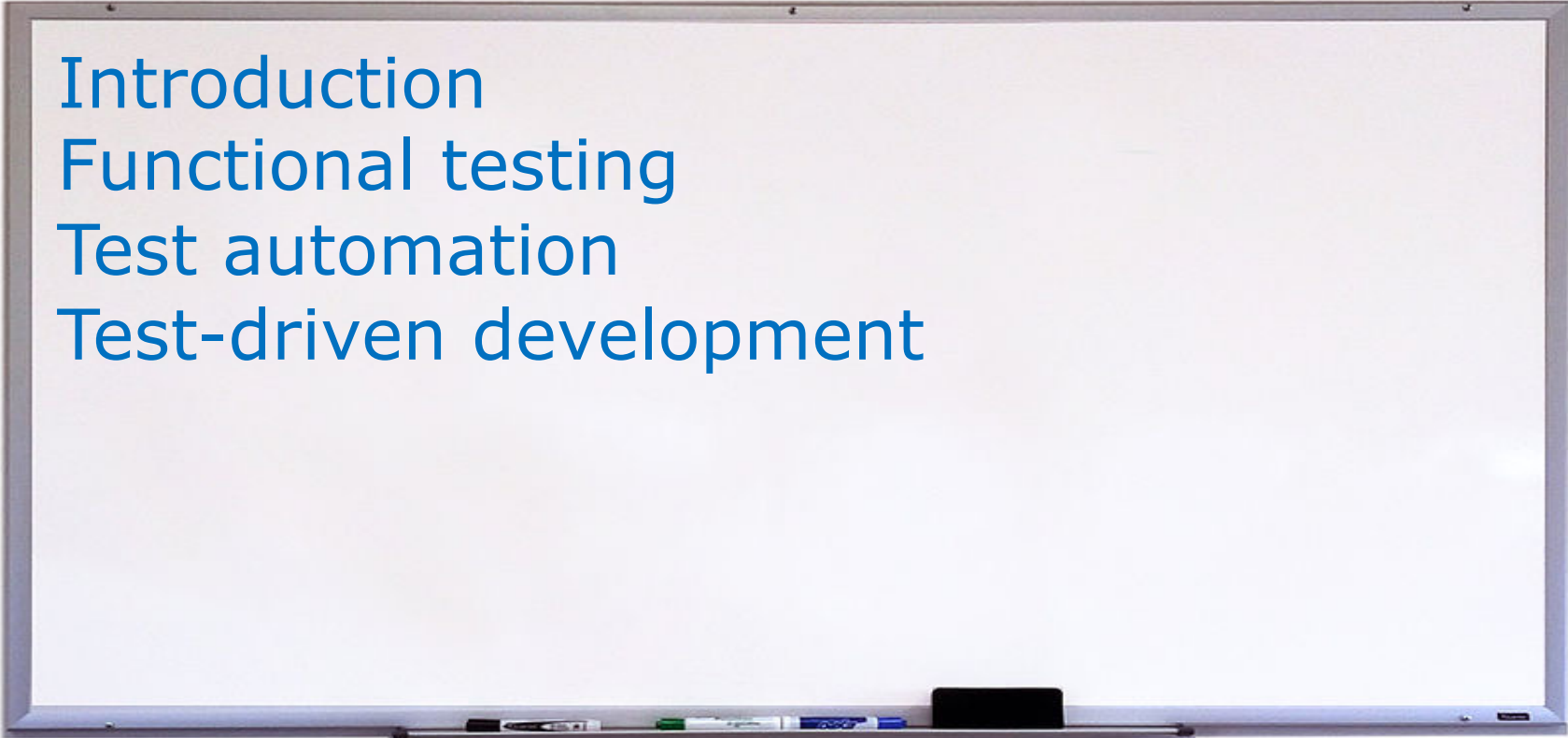
System testing involves testing system as a surrogate user

Perform sequences of user actions

Manual system testing boring and error prone

Testing tools record series of actions and automatically replay them



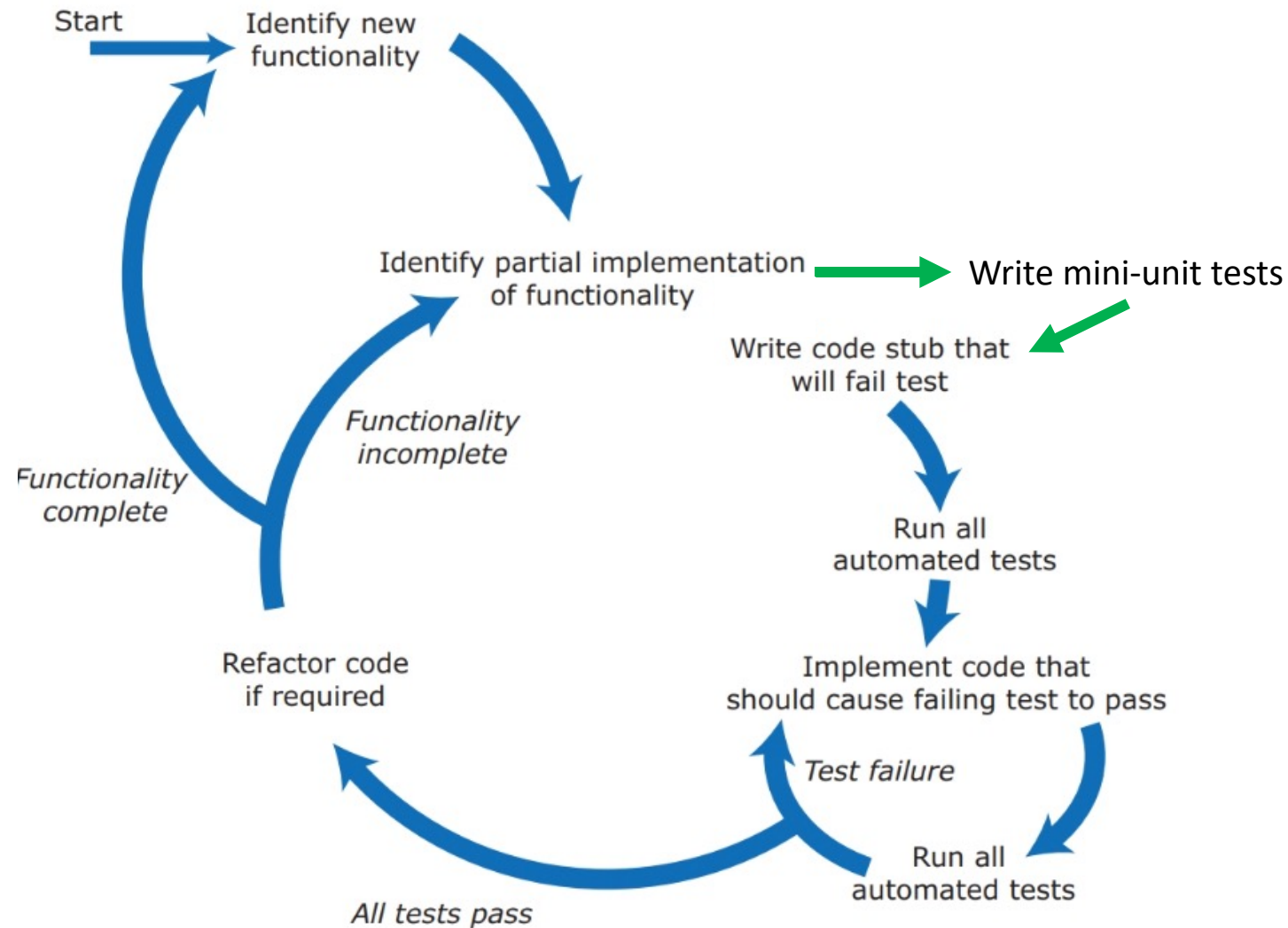
A whiteboard with a silver frame and a white surface. On the left side, there is a list of four topics written in blue text. At the bottom of the whiteboard, there is a black eraser and several markers (black, green, blue, and white) lying horizontally.

Introduction
Functional testing
Test automation
Test-driven development

Test-driven development



"First write executable test, then write the code" [XP]



Test-driven development

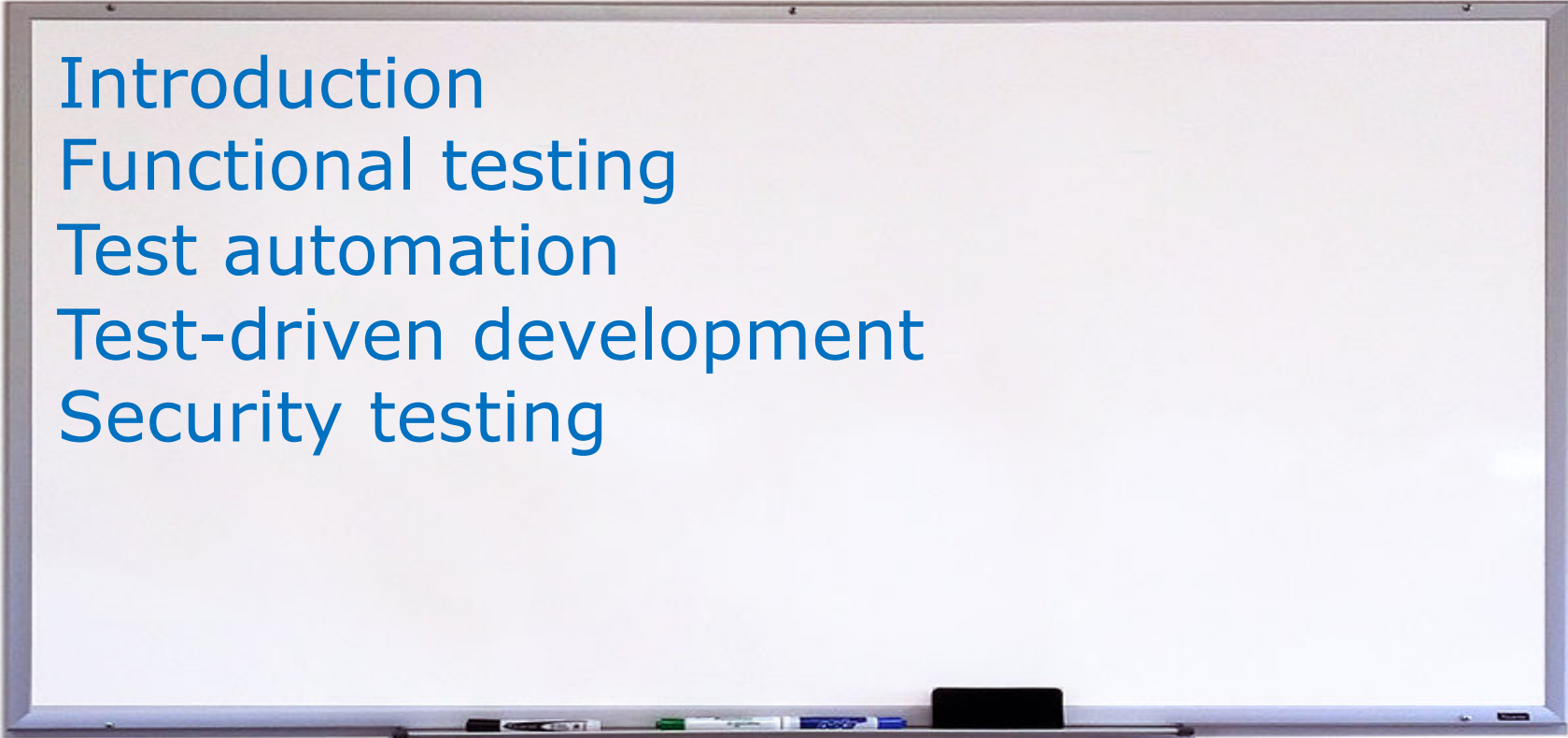
Pros

- Systematic approach: tests clearly linked to code sections, no untested sections
- Tests help understanding program code
- Simplified, incremental debugging
- (Arguably) simpler code



Cons

- Difficult to apply TDD to system testing
- TDD discourages radical program changea
- TDD leads you to focus on the tests rather than on the problem you are trying to solve
- TDD leads you to think too much about implementation details rather than on overall program structure
- Hard to write "bad data" tests

A whiteboard with a silver frame and a white surface. On the left side, a list of five software testing topics is written in blue marker. At the bottom of the whiteboard, there is a black eraser and several markers in black, green, and blue.

Introduction
Functional testing
Test automation
Test-driven development
Security testing

Security testing

Objectives:

- Find vulnerabilities that an attacker may exploit
- Provide convincing evidence that system is sufficiently secure

Finding vulnerabilities harder than finding bugs

- You must test for something that software should NOT do (potentially infinite number of tests)
- Normal functional tests may not reveal vulnerabilities
- The software stack (OS, libraries, dbs, ...) on which your product depends may contain vulnerabilities

Security testing

Comprehensive security testing requires specialist knowledge (e.g. involve external specialists for penetration testing, expensive)

Adopt a risk-based approach

- identify main security risks to your product
- develop tests to demonstrate that the product protects itself from these risks

Possible to automate some of these tests, others inevitably require manual checking of behaviour/files

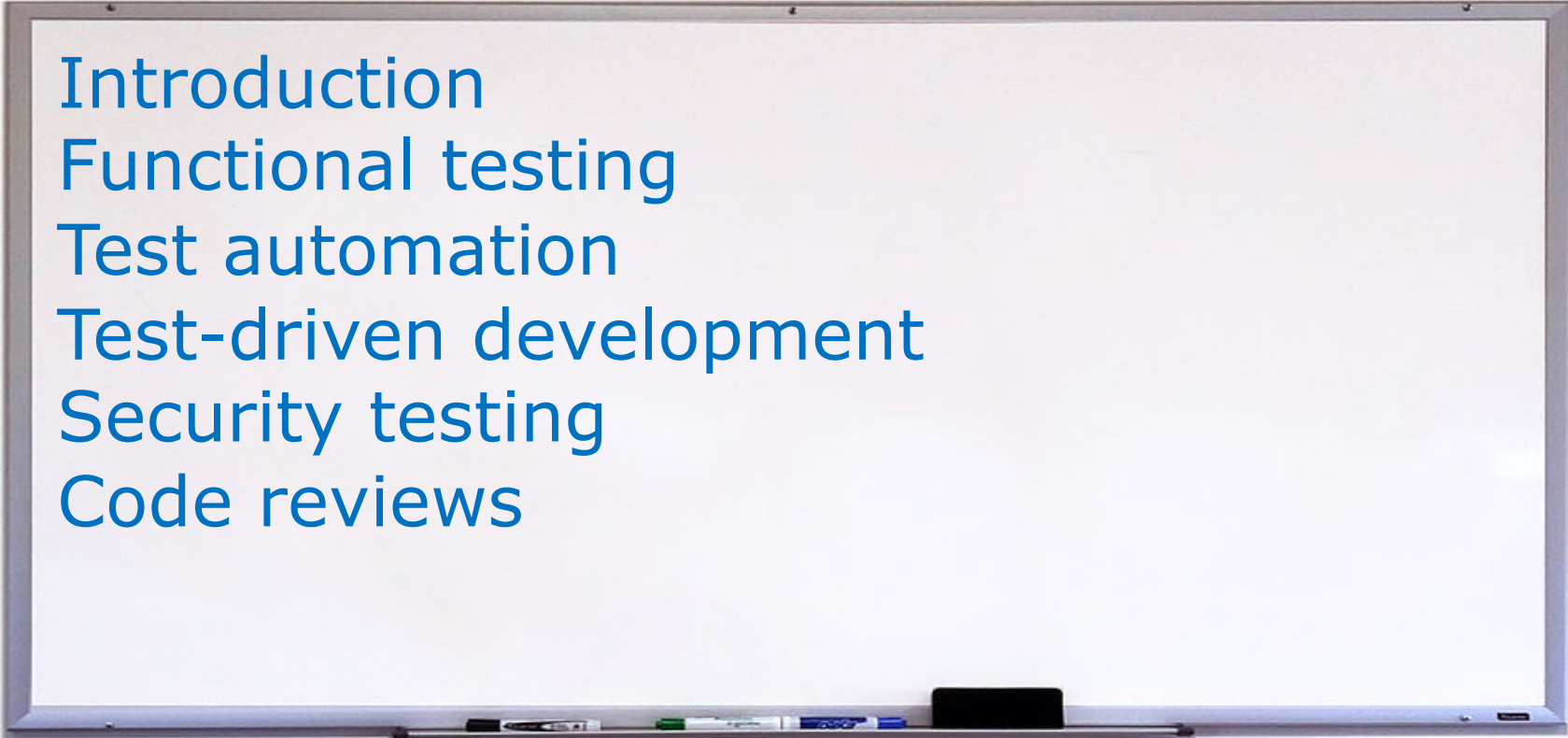
Examples of security risks

- Unauthorized attacker gains access to a system using authorized credentials
- Authorized individual accesses resources that are forbidden to that person
- Authentication system fails to detect unauthorized attacker
- Attacker gains access to database using SQL poisoning attack
- Improper management of HTTP sessions
- HTTP session cookies are revealed to an attacker
- Confidential data are unencrypted
- Encryption keys are leaked to potential attackers

Security testing

Mindset: when testing security, you need to think like an attacker rather than a normal end-user

- deliberately try to do the wrong thing
- repeat actions several times

A whiteboard with a silver frame and a white surface. On the left side, a list of software testing topics is written in blue marker. At the bottom of the whiteboard, there is a black eraser and several markers in black, green, and blue.

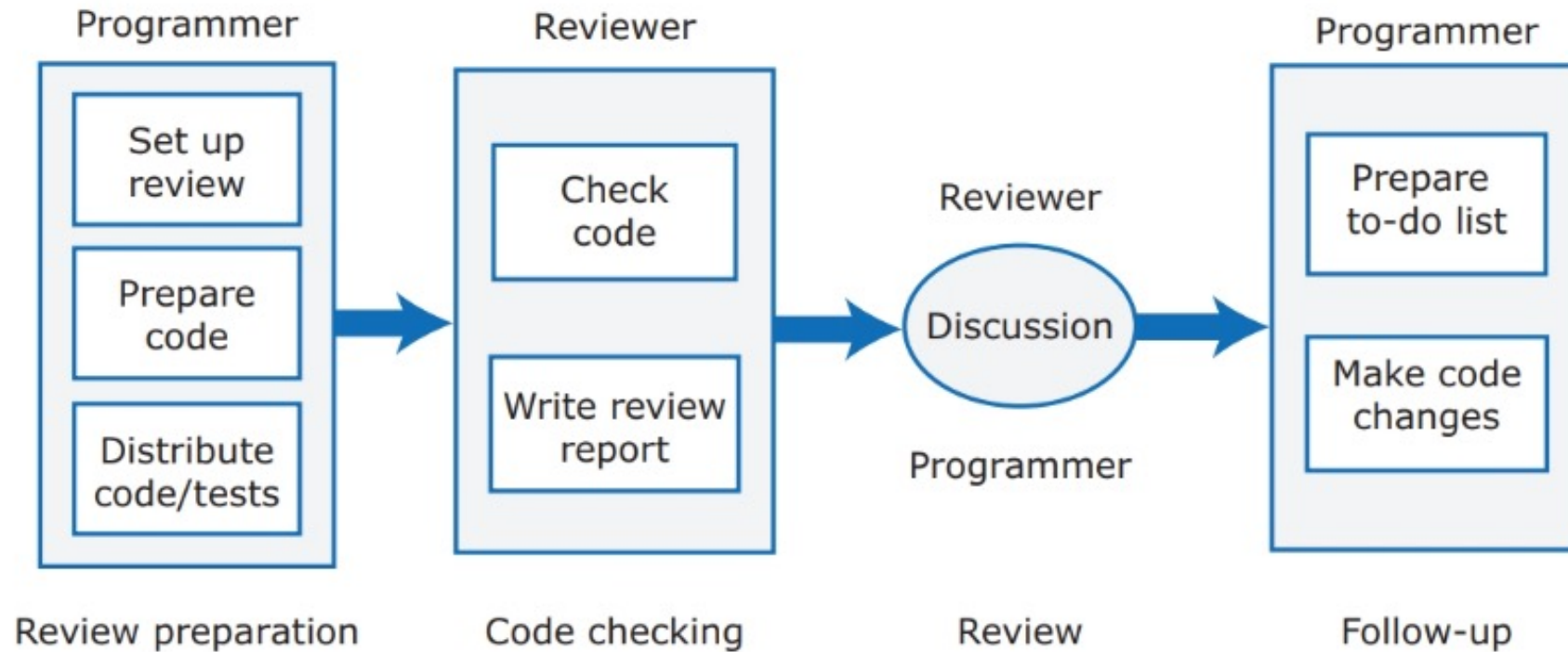
Introduction
Functional testing
Test automation
Test-driven development
Security testing
Code reviews

Limitations of testing

1. You test code against your understanding of what that code should do. If you have misunderstood the purpose of the code, then this will be reflected in both the code and the tests.
2. Testing may not provide coverage of all the code you have written. (TDD shifts the problem to code incompleteness).
3. Testing does not really tell you anything about other attributes of a program (e.g. readability, structure, evolvability).

Code reviews

Code reviews complement testing



Code reviews

Often a single code reviewer is used (part of same DevOps team, or not)

Reviewer can also comment on readability and understandability of code

Review session should focus on 200-400 lines of code

Checklist often used, e.g.

- Are meaningful variables and function names used? (General)

- Have all data errors been considered and tests developed for these? (General)

- Are all exceptions explicitly handled? (General)

- Are default function parameters used? (Python)

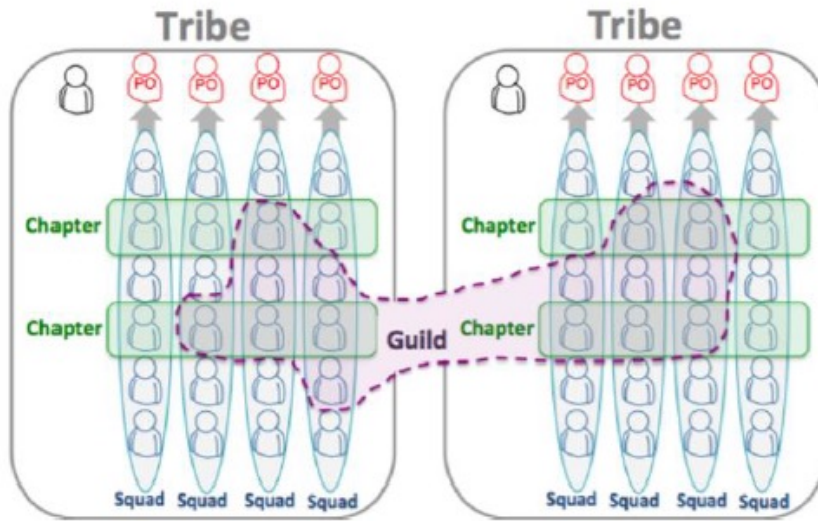
- Are types used consistently? (Python)

- Is the indentation level correct? (Python)

Reviews can be triggered by commits on shared repositories

Several code review tools available, often used together with messaging systems like Slack

Code reviews (example)



Chapters are team members working within a special area (e.g., front office developers, back office developers, database admins, testers).

A **guild** is a community of members across the organization with shared interests (e.g., web technology, test automation), who want to share knowledge, tools code and practices.

Chapters (sometimes guilds) do **code reviews** for squads. Two «+1» required to merge.

Reference



Chapter 9 – Testing

LoL (?)

“Pay attention to zeros. If there is a zero, someone will divide by it.”

“Testers don’t break software, software is already broken.”