UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

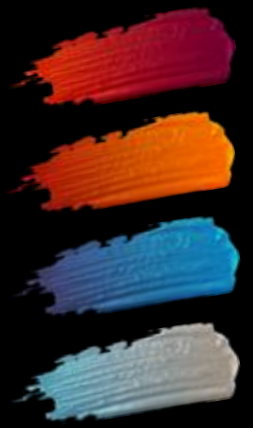
SERLAB
Software Engineering Research

# INTEGRAZIONE E TEST DI SISTEMI SOFTWARE

## Corso di Laurea in Informatica e Tecnologie per la Produzione del Software

Prof.ssa Azzurra Ragone

Dipartimento di Informatica - Università degli Studi di Bari
Via Orabona, 4 - 70125 - Bari
Tel: +39.080.5443270 | Fax: +39.080.5442536
serlab.di.uniba.it



# What is Software Testing?

# Testing: see if the software behaves as expected…





# Testing is a broad term

- Unit test -> testing a small piece of code
- Acceptance testing -> customer validation of a large information system
- Monitoring at run-time
- When?
- …

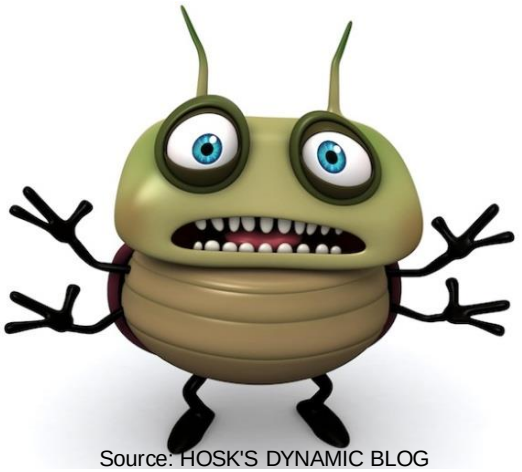
# Different objectives

- Deviations from **users' requirements**
- Evaluating **robustness** to stressful load conditions
- Evaluating robustness to malicious inputs
- Measuring **performance**
- Measuring **usability**
- Operational **Reliability**
- …



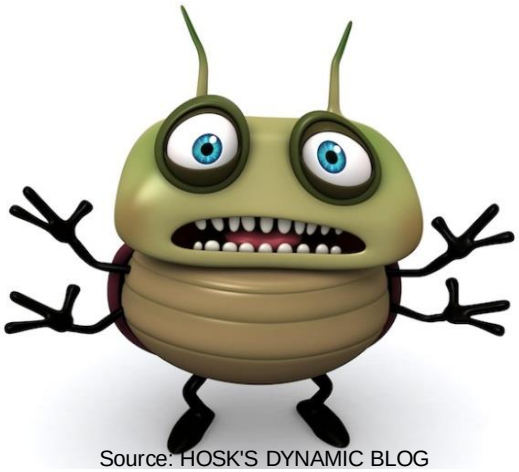


# Testing is also an effective way to find bugs



Source: HOSK'S DYNAMIC BLOG

*"Program testing can be used to show the presence of bugs, but never to show their absence"* **Dijkstra**



Source: HOSK'S DYNAMIC BLOG

# Question to ask

- What tests should you write?
- Are they enough?
- Why these tests?
- …
- What if you change the system?

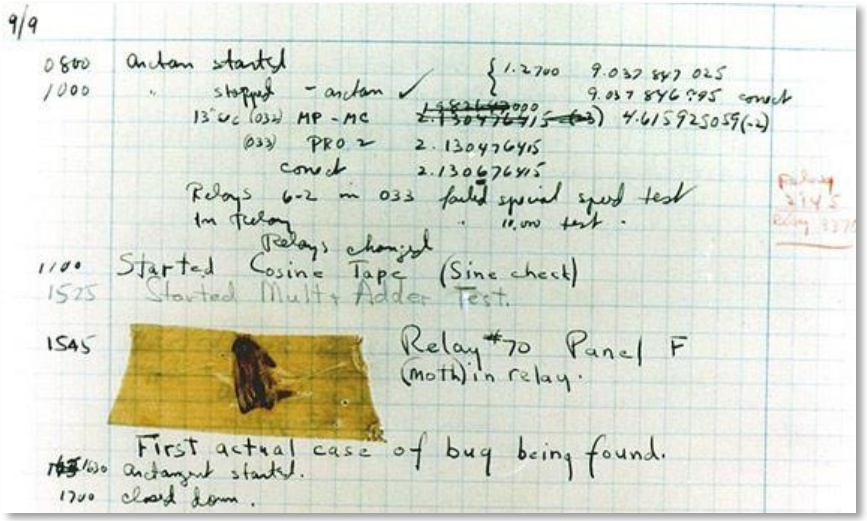
# Who is a Software Tester?

**To be an effective developer,**
**you must become an effective software tester**

**Developers are responsible not only for building software, but also for the quality of the software they produce…**

**Developers are responsible not only for building software, but also for the quality of the software they produce…**
**…tests help you with that responsibility**

# What is this?




- In September **1947**, the Mark II computer broke down at Harvard University
- The cause was a moth that had entered the computer and had shorted out a relay
- The technicians recorded the incident in their notebook and wrote: "*First actual case of bug being found*"
- This is the story of the "*Moth-er of all bugs*"

Slide courtesy of Simone Romano

# Why software testing?


- **Heartbleed** is a security bug in the **OpenSSL cryptography library**
- It was introduced into the code in 2012 and publicly disclosed in 2014
- Half a million of "safe" websites (according to certification Authorities) were actually "unsafe"

Slide courtesy of Simone Romano

# Heartbleed


xkcd comic from 2014

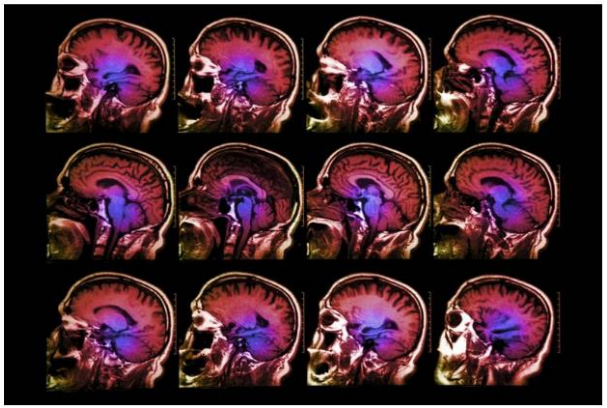
# Why software testing?

## Bug in fMRI software calls 15 years of research into question

Popular pieces of software for fMRI were found to have false positive rates up to 70%

By **EMILY REYNOLDS**

*Wednesday 6 July 2016*



Slide courtesy of Simone Romano

# Why software testing?

BUSINESS NEWS SEPTEMBER 9, 2016 / 6:19 PM / 3 YEARS AGO

## GM recalls 4.3 million vehicles over air bag-related defect

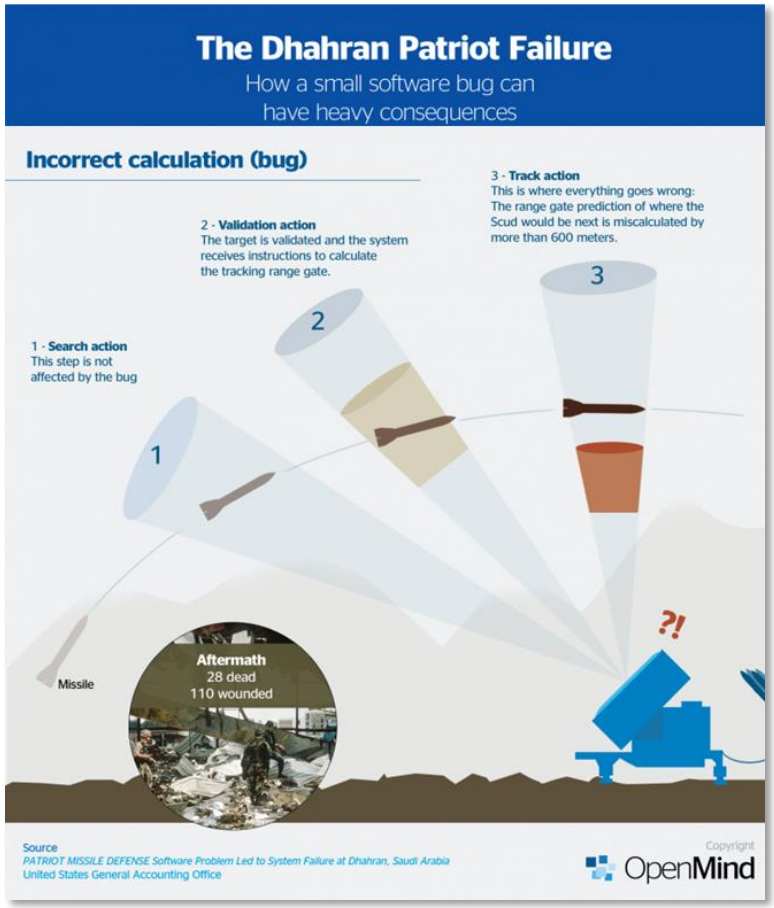
David Shepardson 3 MIN READ

(Reuters) - General Motors Co (GM.N) said on Friday it will recall nearly 4.3 million vehicles worldwide due to a software defect that can prevent air bags from deploying during a crash, a flaw already linked to one death and three injuries.


- Takata airbag bug caused the death of more than 100 people

Slide courtesy of Simone Romano

# Why software testing?




- In 1991, an Iraqi missile hit the US base of Dhahran
- The base's antiballistic system, Patriot, failed to launch because of a bug
- The system's internal clock drifted by milliseconds (a delay of ⅓ of a second after 100 hours)
- This "micro-delay" led to a "600 meter" error
- A Scud travels at about 1,676 meters per second, and so travels more than half a kilometer in this time.

Slide courtesy of Simone Romano

# Why software testing?


- Many software systems handled the "**year**" variable with only **two digits** because writing "19" was deemed a waste of memory
- All good until the turn of the century
- Billions of dollars spent to fix the **Millennium** bug

Slide courtesy of Simone Romano

# Why software testing?


- **Blue Screen** of Death during Windows 98 presentation

Slide courtesy of Simone Romano

**Computers can run hundreds of tests in a second…is it enough?**





# Test automation

Automation executes tests that a developer designed

If tests are not good or do not cover parts of the code that contain bugs, they are less useful

# Test automation

Test case design:
engineer the test cases
*(Creativity is here!)*

vs

Test case execution:
automate them with some code (e.g. JUnit)

# Principles of software testing

1. Exhaustive testing is impossible (prioritize)
2. Knowing when to stop testing (time, money, adequacy criteria)
3. There is no silver bullet (the pesticide paradox)
4. Bugs happen in some places more than others (not unif. distr.)
5. Tests cannot ensure software is 100% bug-free (Dijkstra)
6. Context is king
7. Verification is not Validation

Inspired by Software Testing Qualifications Board (ISTQB)

# Validation vs Verification

Validation is when the software system meet the user's needs and fulfill requirements.

Verification is checking the consistency of an implementation with respect to a specification.

A requirement analysis describe the problem we want to solve.

The specification describes the solution.





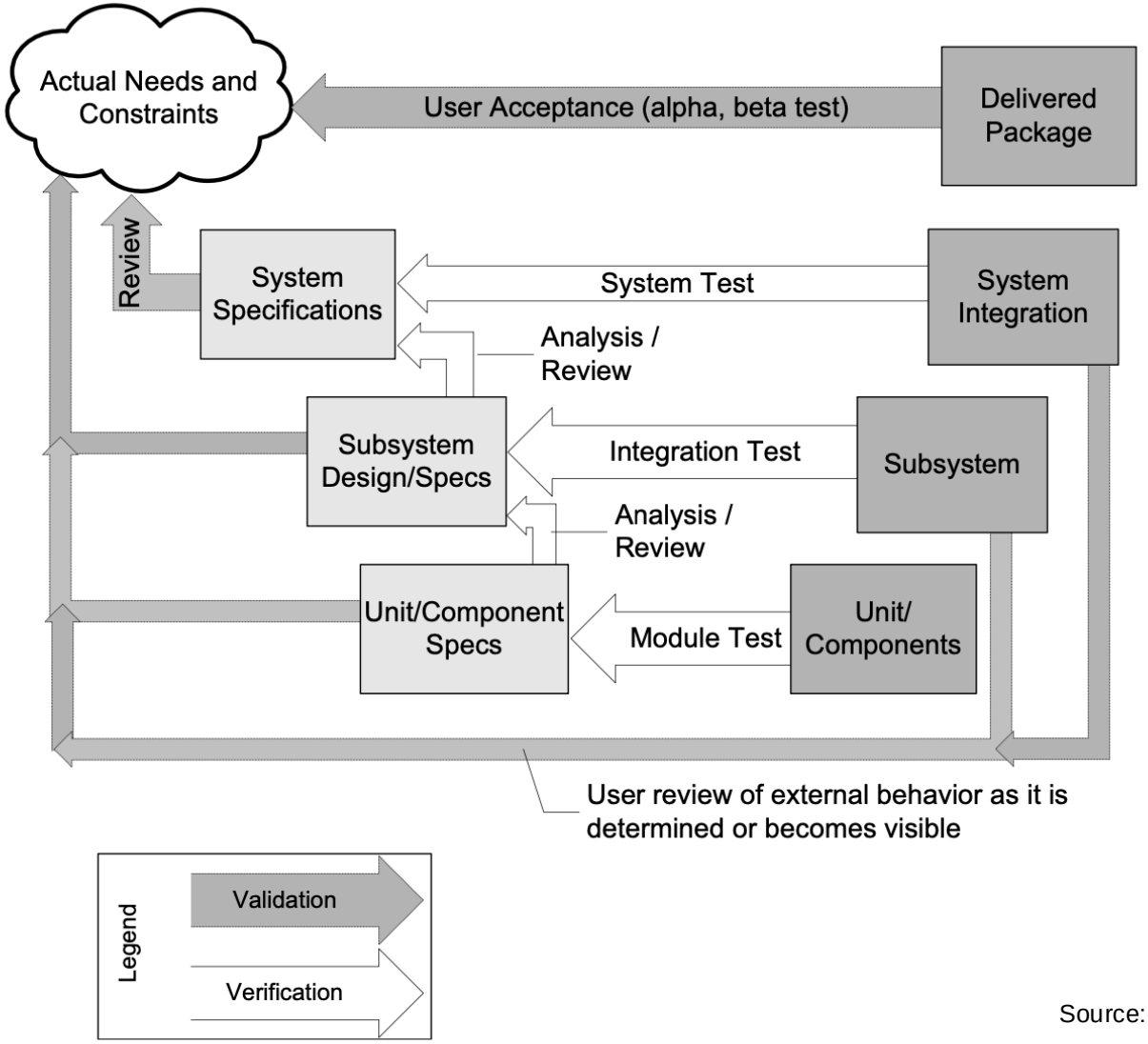
# Verification is not Validation

Verification is about having the *system right* (is about implementation)

vs

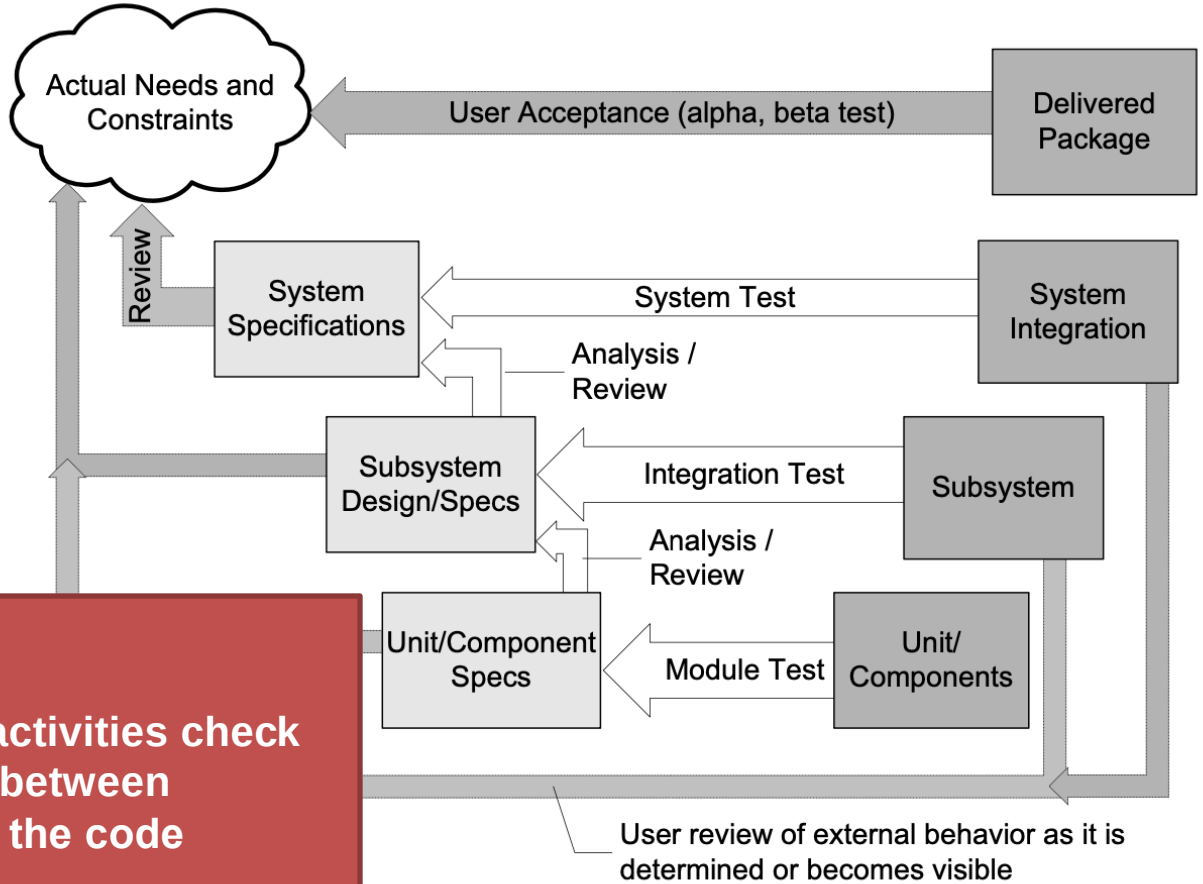
Validation is about having the *right system* (is about user requirements)

# Verification is not Validation




Source: Pezzè & Young book

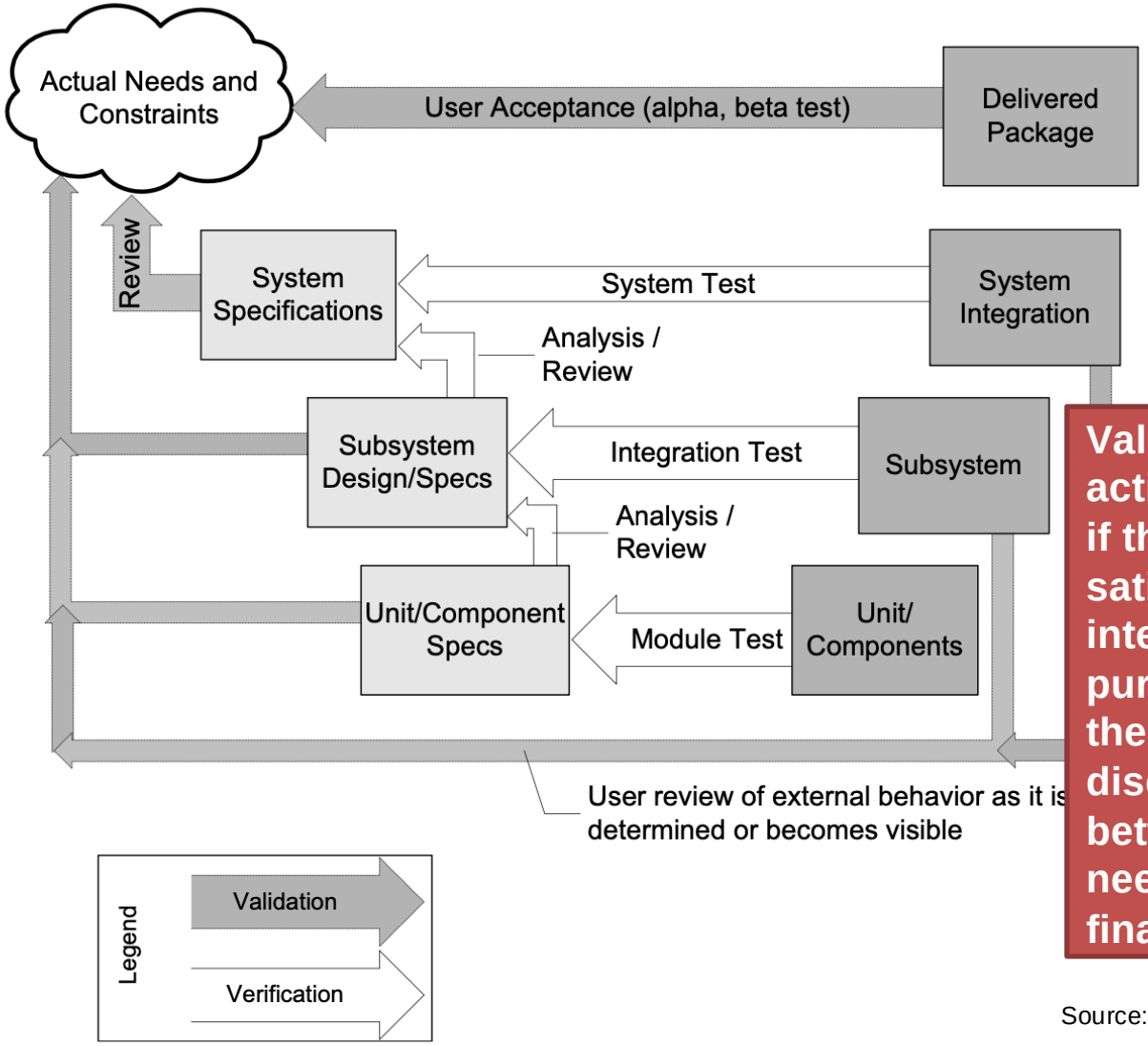
# Verification is not Validation




**Verification activities check consistency between "Specs" and the code**

Source: Pezzè & Young book

# Verification is not Validation

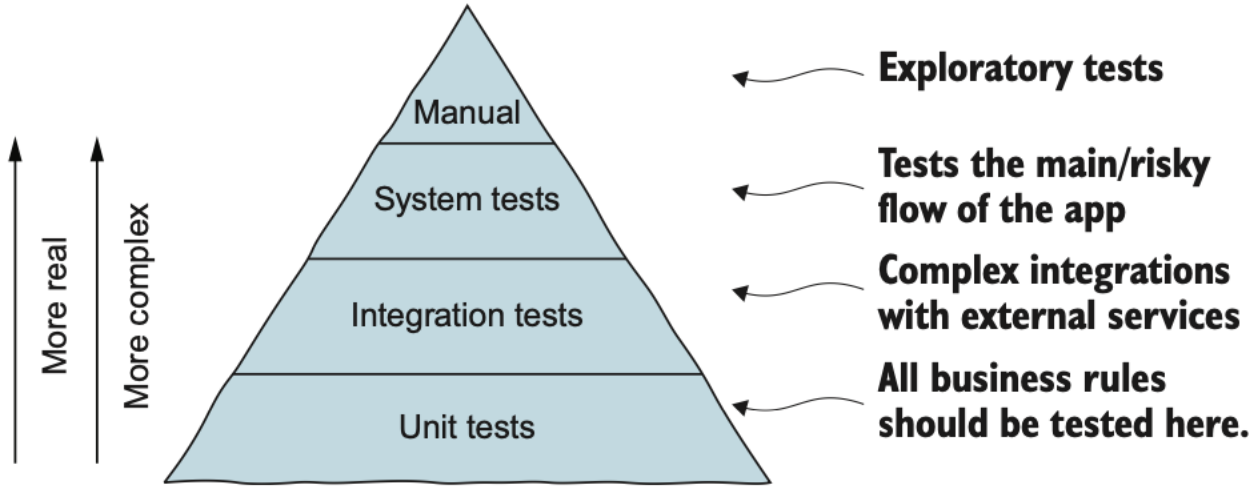



**Validation activities check if the system satisfies its intended purpose, and if there are discrepancy between actual need and the final product**

Source: Pezzè & Young book

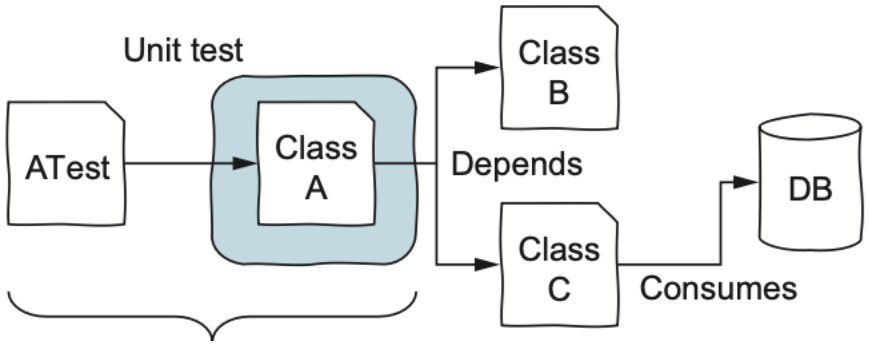
# The testing pyramid

- The size of the slice in the pyramid represents the relative number of tests to carry out at each test level



# The testing pyramid: Unit tests

Unit testing means testing a (small) set of classes that have no dependency on external systems (such as databases or web services) or anything else you cannot fully control.



# The testing pyramid: Unit tests

PLUS

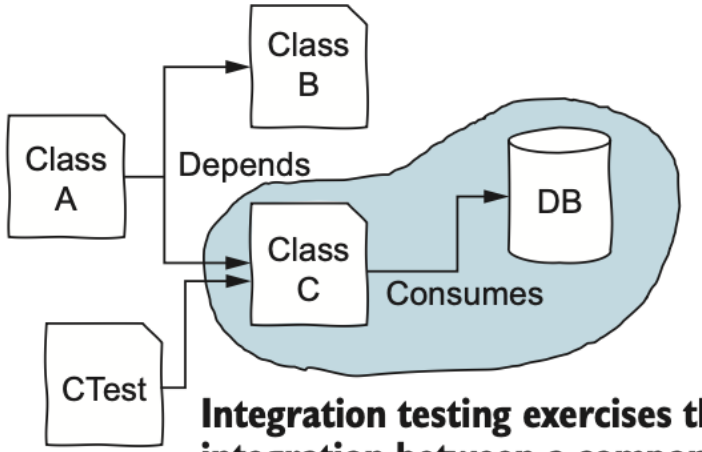
- Test **single feature** of the software: ex. a method, a class, etc.
- Test units in **isolation**
- Unit tests are **fast** and give **feedback** (improved code design)
- Unit tests **assert** different outputs given the inputs
- Unit test are **easy** to write (wrt DB, frontends, web services, etc.)

MINUS

- UTs do not represent the real execution of a software system
- Some bugs cannot be caught at the unit test level (frontend+backend, multithread, etc.)

# The testing pyramid: Integration tests

Integration testing aims to test multiple components of a system together, focusing on the *interactions* between them, as testing components in isolation sometimes is not enough (DB, WS, files r/w)

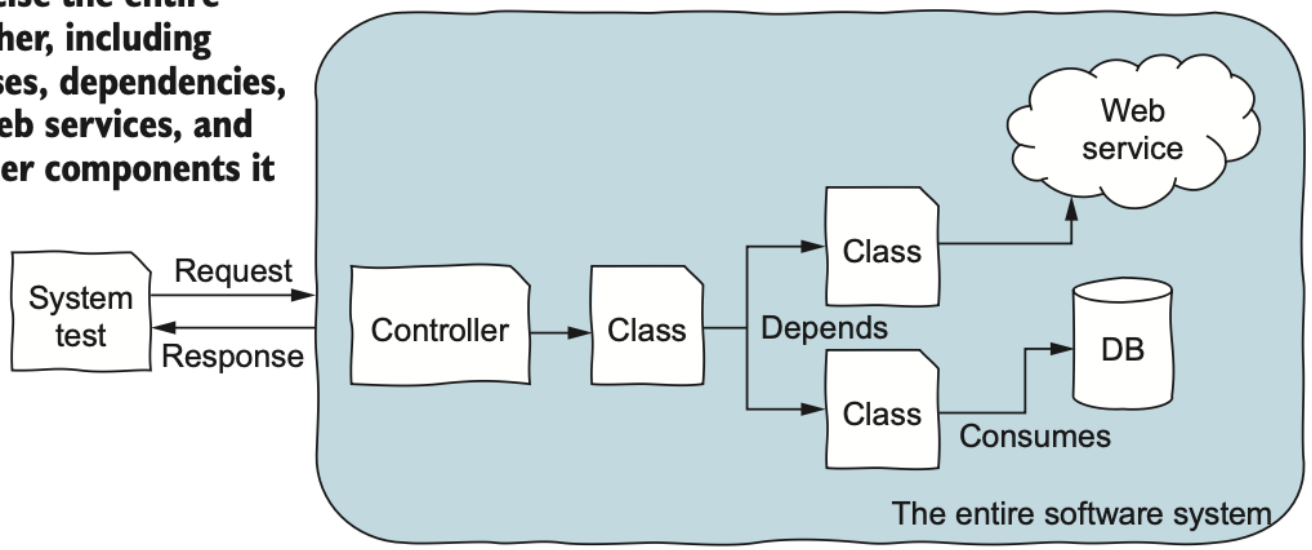



**Integration testing exercises the integration between a component of your system and some external component (e.g., a database).**

# The testing pyramid: System tests

System testing run the entire system together, with DB, frontend apps, etc.. We do not care how the system works from the inside (how it is implemented). Ensure the product is tested from the end user perspective.

**When system testing, you want to exercise the entire system together, including all of its classes, dependencies, databases, web services, and whatever other components it may have.**



# The testing pyramid: System tests

PLUS

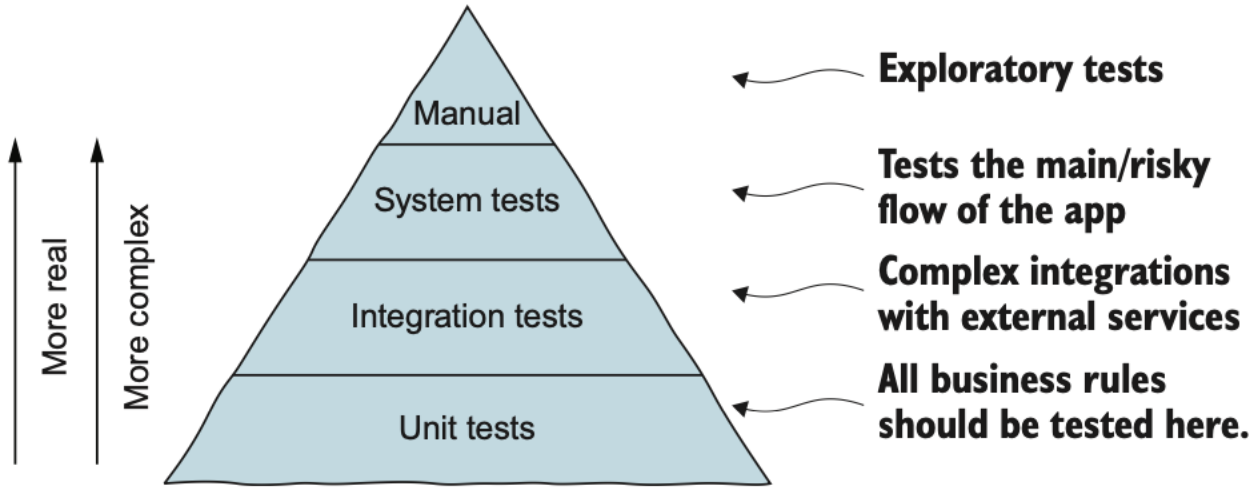
- They are **real**: they perform actions similar to the ones preformed by the final user

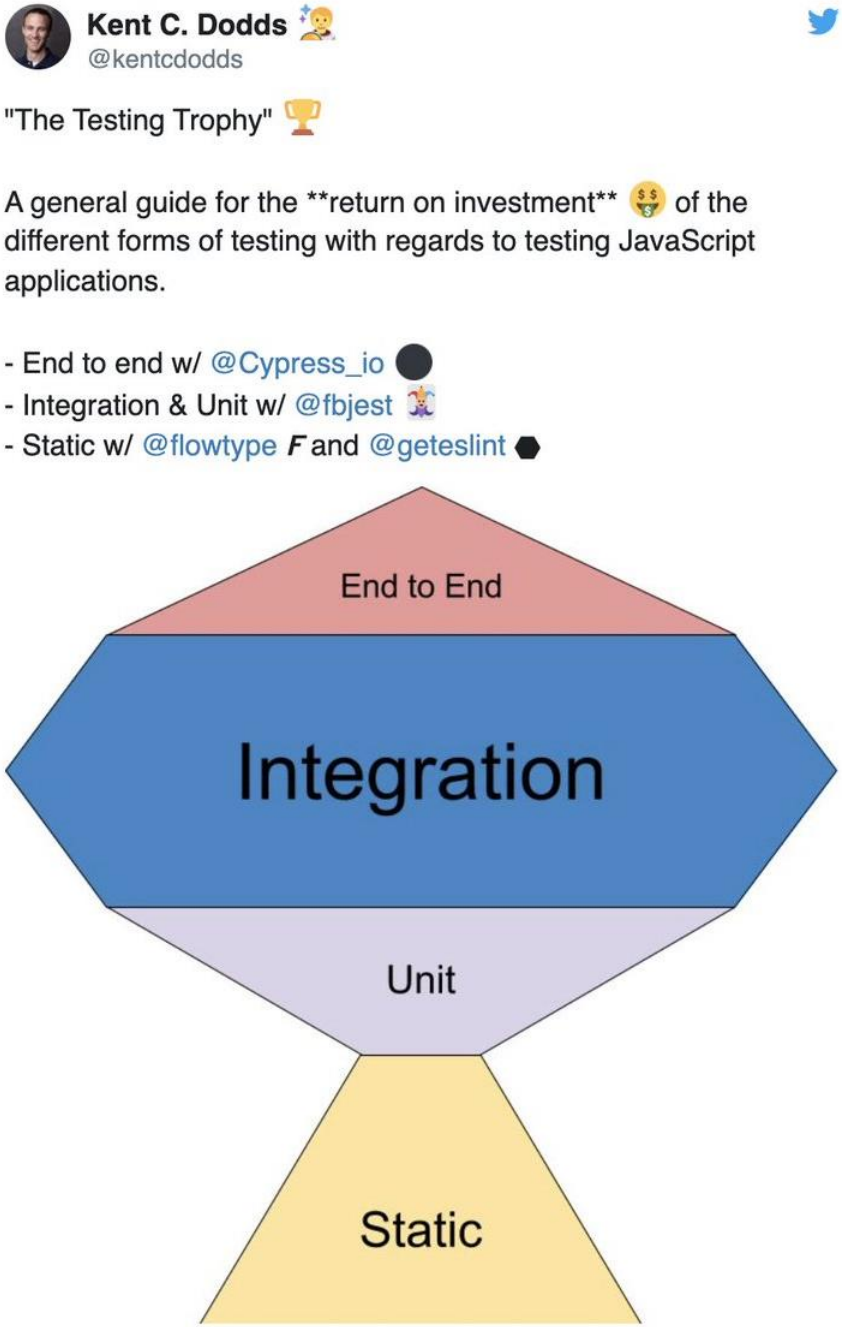
MINUS

- STs are slow
- They require a **complex setup** and additional code to automate the tests
- STs are sometime **unreliable** as linked to variations out of our control

# The testing pyramid

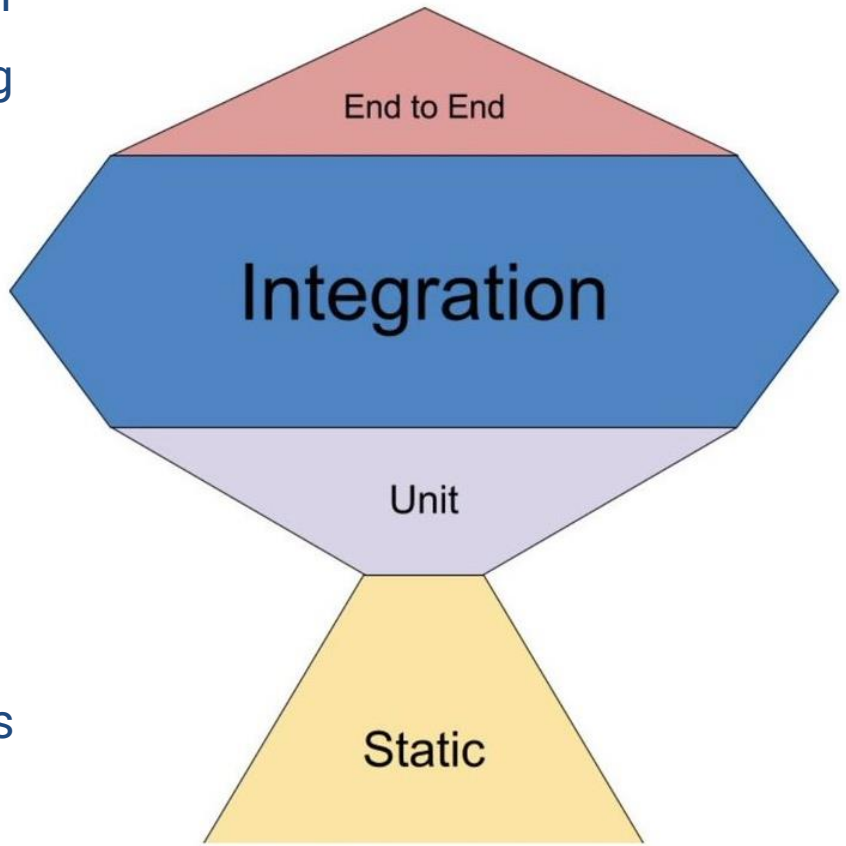
- STs and ITs are more expensive and hard to set up
- STs are very costly, difficult to write and slow to run




Kent C. Dodds
@kentcdodds
"The Testing Trophy"
A general guide for the **return on investment** of the different forms of testing with regards to testing JavaScript applications.
- End to end w/ @Cypress_io
- Integration & Unit w/ @fbjest
- Static w/ @flowtype F and @geteslint
End to End
Integration
Unit
Static
556 11:53 AM - Feb 6, 2018

# The testing trophy

- In many software systems, most of the complexity is in integrating components, e.g.:
  - highly-distributed microservices architecture
  - database-centric information systems




There is not a «fix-it-all» solution, as usual, context is king!

**This course is about:**
**Test your software in an EFFECTIVE and SYSTEMATIC way**





# Effective and systematic

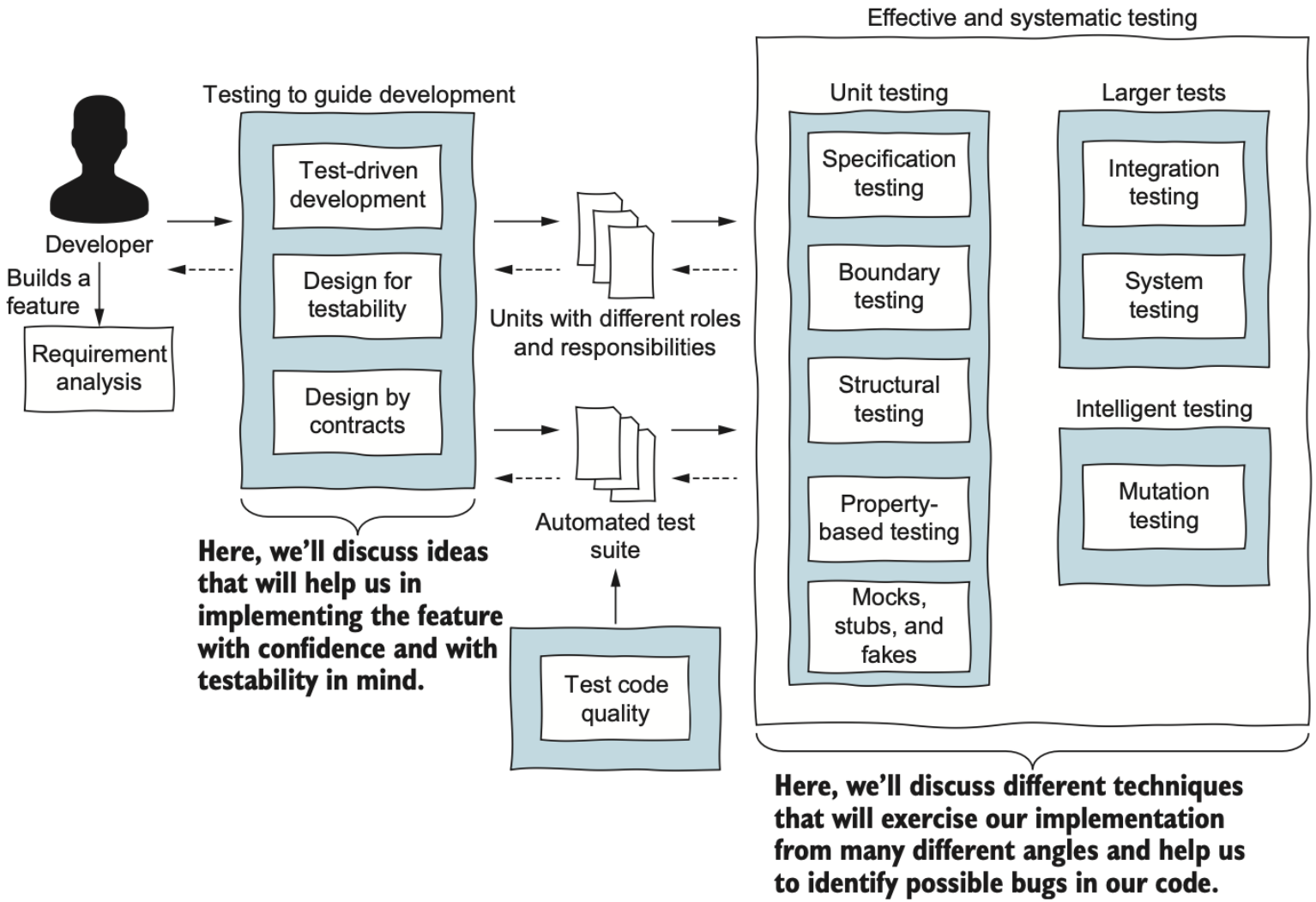
## Effective-> Write the right tests:

maximize the number of bugs found while minimizing the effort required to find the bugs (trade-off).

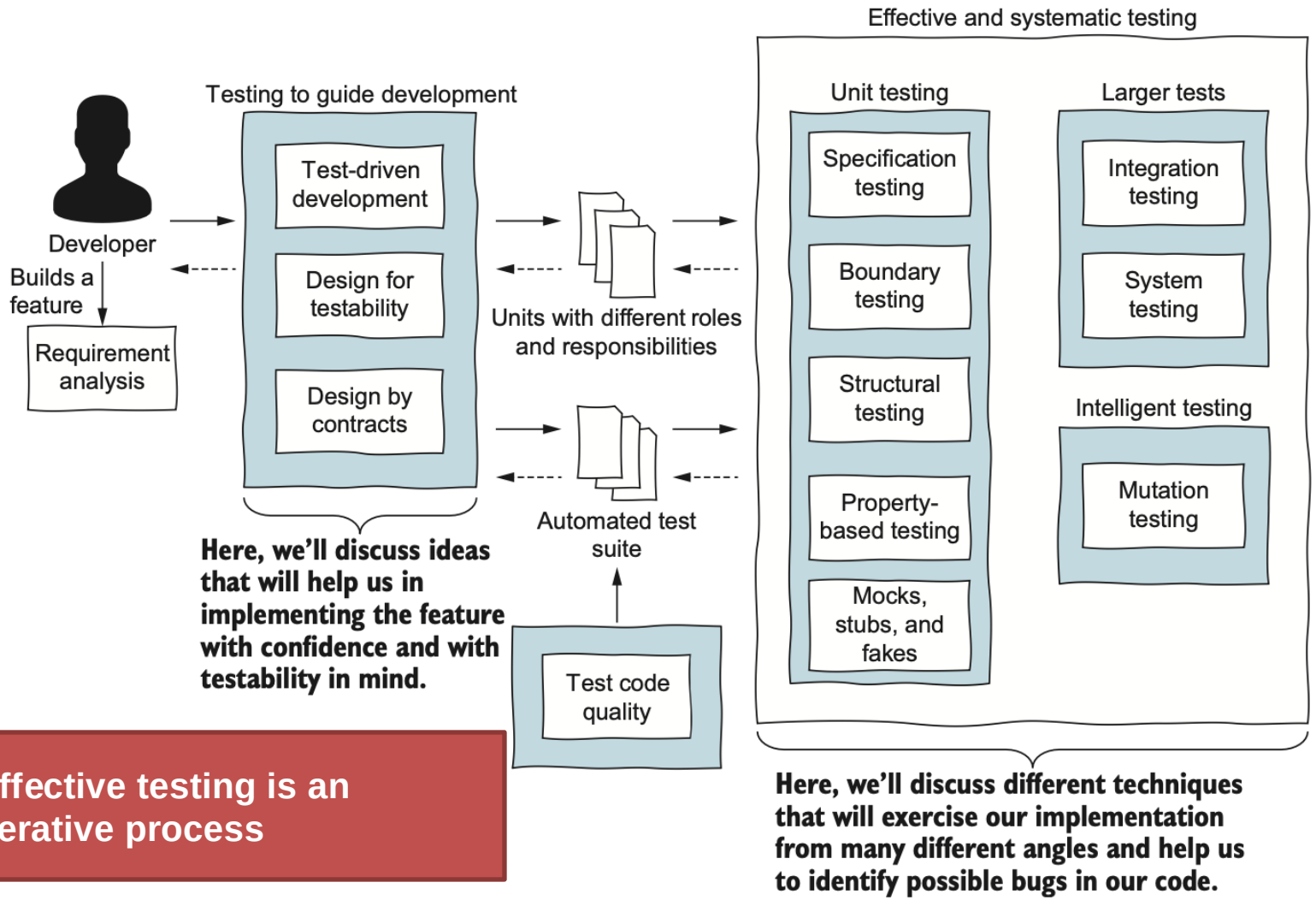
## Systematic:

for a given piece of code, any developer should come up with the same test suite.

# Effective and systematic testing (workflow)



# Effective and systematic testing (workflow)



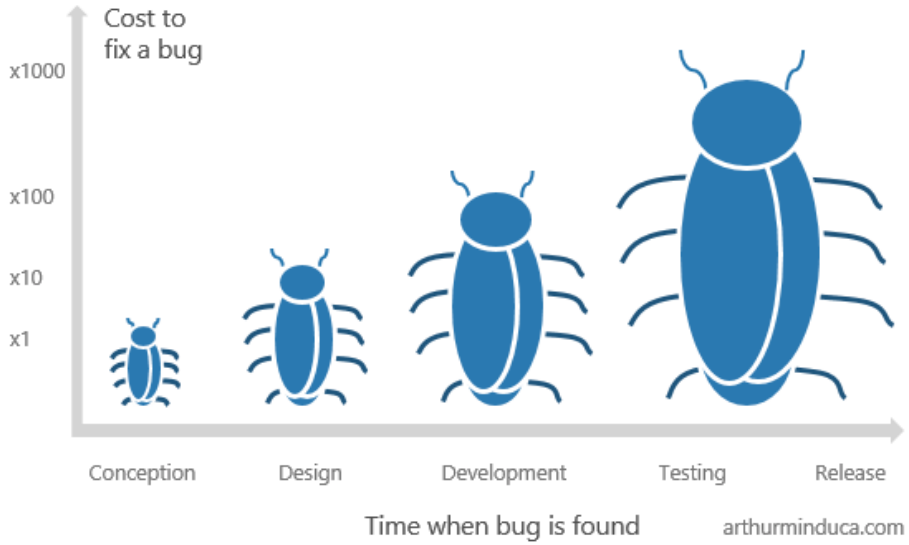

**Effective testing is an iterative process**

# The cost of testing

- The cost of bugs in production outweighs the cost of prevention (ex. Shopping web site)

- *Bugs-customer find it-fix* loop

- The more you write tests, the faster you will be

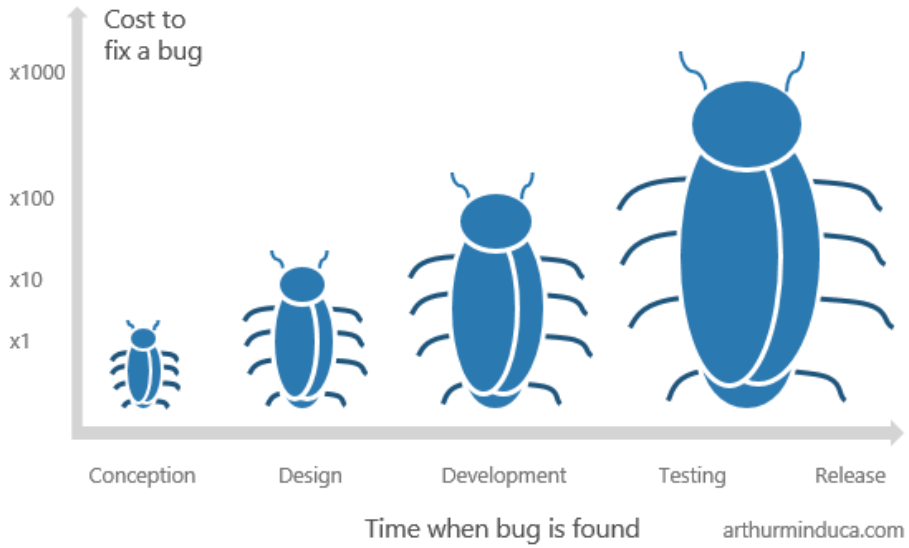


# The cost of bugs

A bug identified **during conception** costs something around zero

For the same bug found **after implementation** or test, the average cost of repair can get to something between 10 and 1000 times more

When customers find this bug in **production environment**, the cost of the problem considers all side effects related to it (reputation)



# Bugs, customers and time

Customers **lose confidence** when they find bugs.

Bugs **frustrate** the customer.

If a developer finds a bug whilst writing the code, she/he can fix it quickly (business logic and code is fresh in her/his mind).

If the bug is found later you need to reread the specs, ask business analyst and relearn the rest of the code.

# Bugs in production

A bug found in **production** takes **much longer** to be fixed:

A customer finds the bugs and raises a **ticket**

The bug is logged by someone

The developer will try and **recreate the bug**. This could lead to more exchanges asking for more information about recreating the bug.

The bug recreated and **fixed** (once the developer understands the code again)

DEV **testing**

**Build** and **deploy** in other environments and testing

# Note sul corso

- Questo corso richiede la partecipazione attiva
- Questo non è un corso su Java
- Le lezioni di teoria e le lezioni di esercitazione sono strettamente interconnesse
- Si consiglia di portare il pc ad ogni lezione, per una migliore fruizione della stessa

# Modalità d'esame

L'esame finale prevede una **prova scritta** con **domande** a risposta **chiusa** e domande a risposta **aperta**.

Inoltre, è prevista la presentazione in aula di un **lavoro di gruppo** (facoltativo):
- Si presenta e si discute in maniera critica il lavoro sviluppato in gruppo;
- Si accertano le capacità espositive dello studente e la capacità di motivare le scelte progettuali.

Il lavoro di gruppo va concordato con il docente e inizia ad essere sviluppato sotto la guida del docente durante le ore di laboratorio. Il lavoro di gruppo è facoltativo e contribuisce, se svolto, alla votazione finale dell'esame conferendo un **bonus da 1 a 3**.

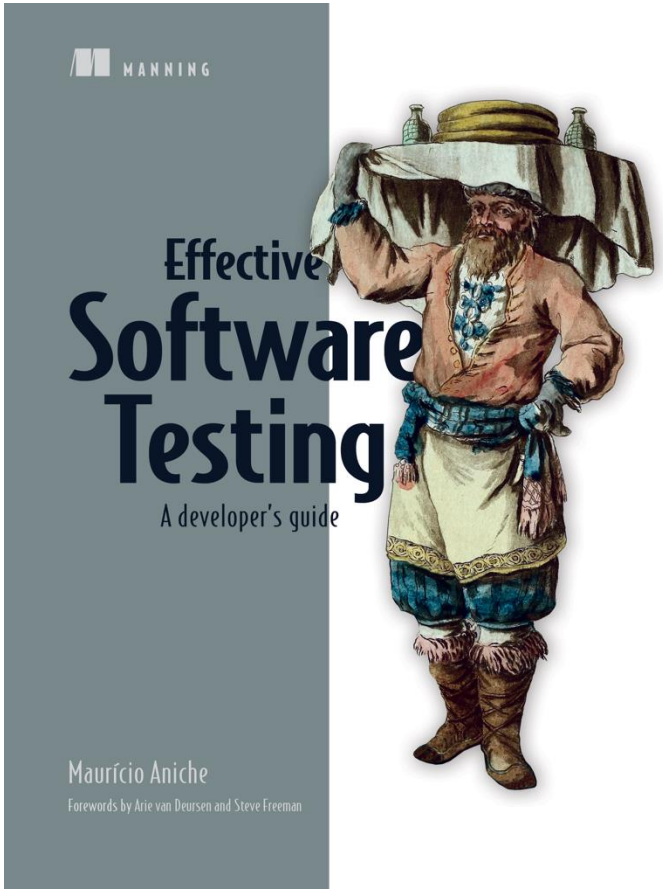
# Testo di riferimento

Effective Software Testing. A developer's guide. Mauricio Aniche. Ed. Manning. (Chapter 2)

Codice sconto del 35% sul prezzo di copertina: "**au35ani**"

https://www.manning.com/books/effective-software-testing

La copia cartacea è disponibile in biblioteca




SLIDES AND
REFERENCE
MATERIAL


**Chiave di accesso: ITSS-2425**

# References

- AssertJ Library: https://assertj.github.io/doc/
- Lambda functions: https://www.w3schools.com/java/java_lambda.asp
- Property-Based Testing in Java Jqwik library: https://jqwik.net/






Azzurra Ragone

Department of Computer Science- Floor VI – Room 616
Email: azzurra.ragone@uniba.it