Software Testing and Reliability

Xiaoyuan Xie 谢晓园

xxie@whu.edu.cn

计算机学院E301

Some Instructions about the Course

- **Regulations**
  - Refer to the .doc file
- **References**
  - Glenford J. Myers , The Art of Software Testing, Second Edition, Wiley Press
  - Myers, G., Software Reliability: Principles & Practices, Wiley Press

Some Instructions about the Course

- **Instructor**
  - Xiaoyuan Xie (xxie@whu.edu.cn)
- **Tutor**
  - Jingdi Xu (505486735@qq.com)

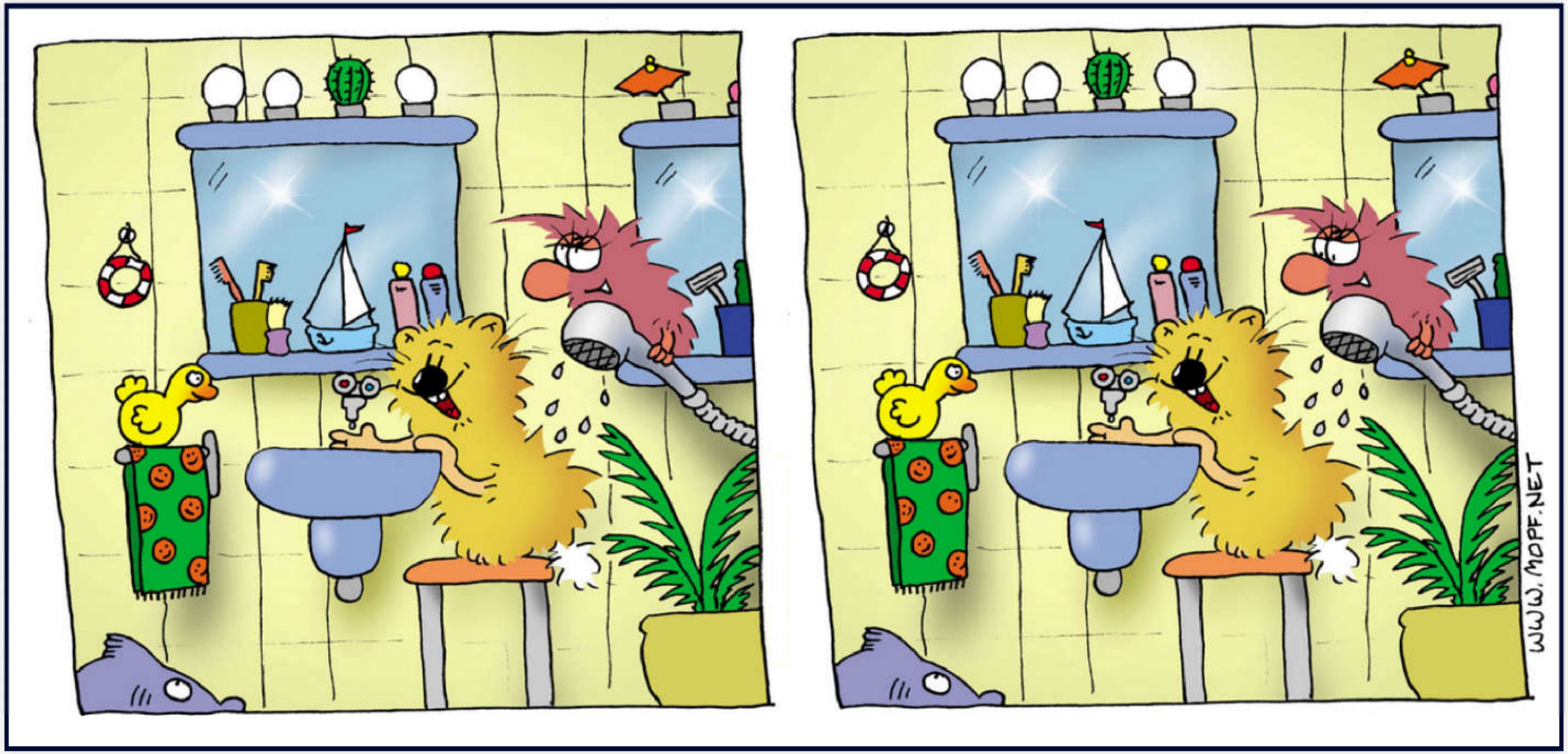
Lecture 2: Introduction

Xiaoyuan Xie 谢晓园
xxie@whu.edu.cn
计算机学院E301

Testing

> *The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.*
>
> —ISTQB®

This is a Test



Different Perspectives, Different Objectives.

Perspective	Objective
Developer	Find defects.
QA	Evaluate quality.
Tester	Check functionality against requirements.
Customer	Verify usability and applicability.

Do Software Testing. Right.

- Testing has to be planned.
- Testing costs easily up to 40 % of a project's budget.
- Testing has to be performed in a reasonable way.
- Testing shall be independent and objective.
- Testing has to be managed.

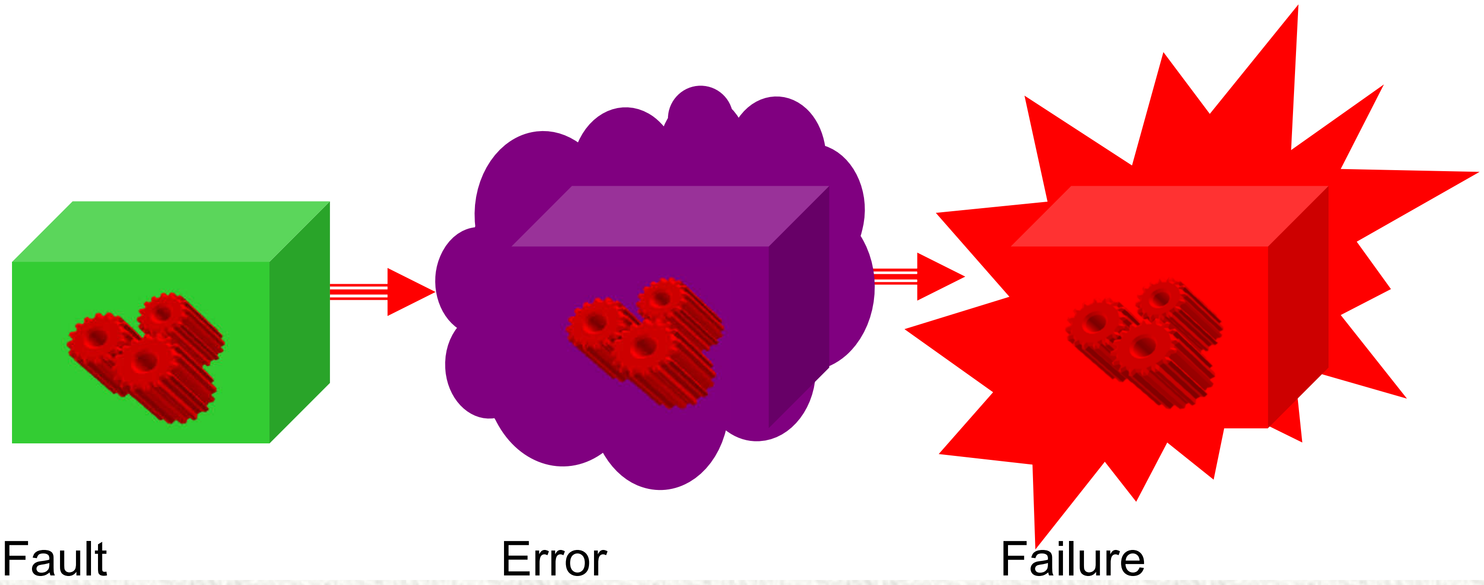
Software testing is a fundamental part of professional software development!

Software Faults, Errors & Failures

- Software Fault : A static defect in the software

- Software Failure : External, incorrect behavior with respect to the requirements or other description of the expected behavior

- Software Error : An incorrect internal state that is the manifestation of some fault



Software Faults, Errors & Failures

M. Roper:

"*When developing software, people make errors, these become faults in the software which then manifest themselves as failures when the software is run*."

"*One error may lead to several different faults, each of which in turn leads to several different failures*."

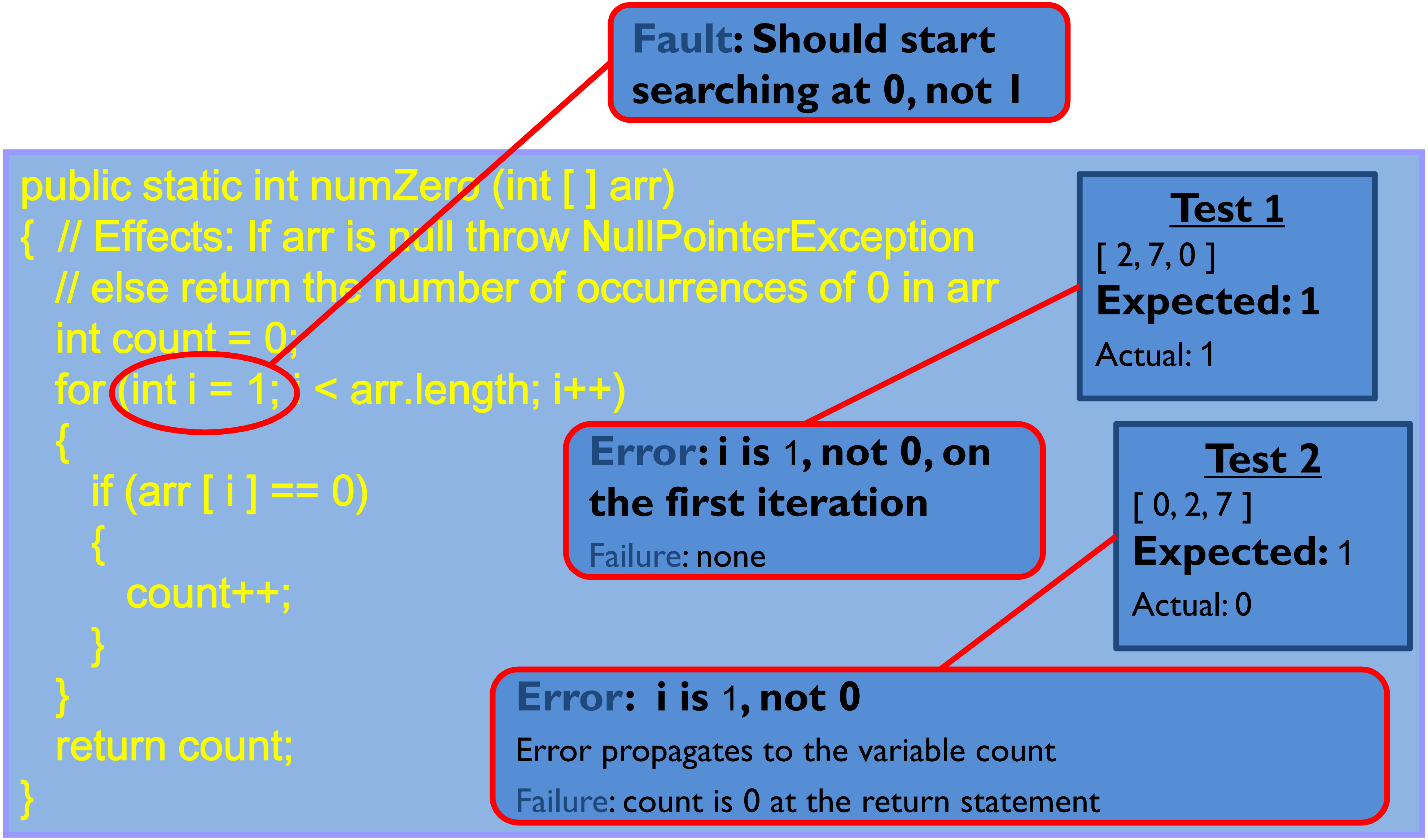
Fault and Failure Example

- A patient gives a doctor a list of symptoms
  - Failures
- The doctor tries to diagnose the root cause, the ailment
  - Fault
- The doctor may look for anomalous internal conditions (high blood pressure, irregular heartbeat, bacteria in the blood stream)
  - Errors

Most medical problems result from external attacks (bacteria, viruses) or physical degradation as we age.

Software faults were there at the beginning and do not "appear" when a part wears out.

A Concrete Example



Fault: Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{  // Effects: If arr is null throw NullPointerException
   // else return the number of occurrences of 0 in arr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
   {
      if (arr [ i ] == 0)
      {
         count++;
      }
   }
   return count;
}
```

Test 1

[2, 7, 0]

Expected: 1

Actual: 1

Error: i is 1, not 0, on the first iteration

Failure: none

Test 2

[0, 2, 7]

Expected: 1

Actual: 0

Error: i is 1, not 0

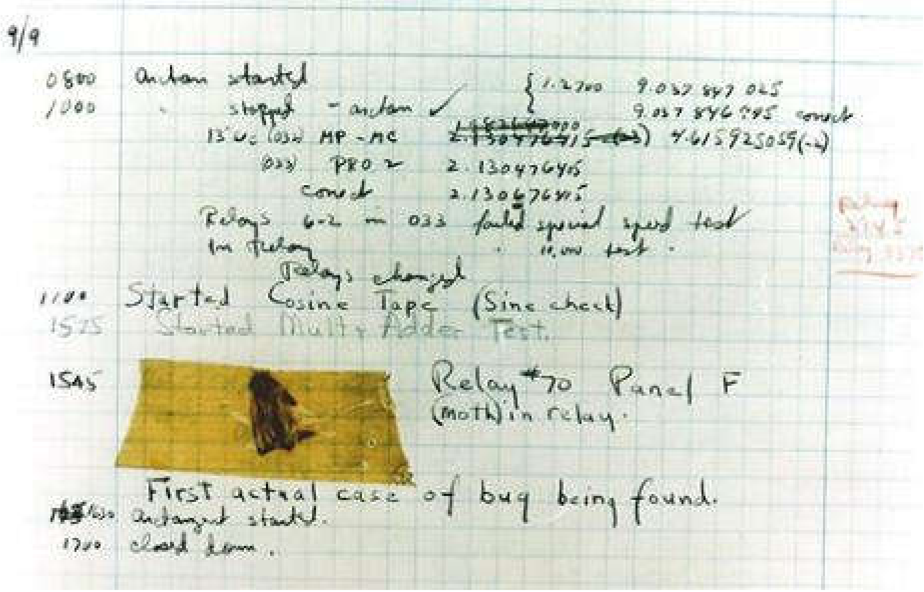
Error propagates to the variable count

Failure: count is 0 at the return statement

The Term Bug

- *Bug* is used informally
- Sometimes speakers mean fault, sometimes error, sometimes failure … often the speaker doesn’t know what it means !
- This class will try to use words that have precise, defined, and unambiguous meanings

The very first recorded computer bug



9/9

0800 Anctan started

1000 " stopped - anctan ✓ {1.2700 9.037 847 025

9.037 846 995 correct

13" UC (032) MP-MC ~~2.130476415~~ (-3) 4.615925059(-2)

(033) PRO 2 2.130476415

correct 2.130676415

Relays 6-2 in 033 failed special speed test

In relay " 10,000 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Multy Adder Test.

1545 Relay #70 Panel F

(moth) in relay.

First actual case of bug being found.

1630 antangent started.

1700 closed down.

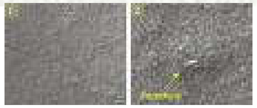
On September 9, 1945, U.S. Navy officer Grace Hopper found a moth between the relays on the Harvard Mark II computer she was working on. In those days computers filled (large) rooms and the warmth of the internal components attracted moths, flies and other flying creatures. Those creatures then shortened circuits and caused the computer to malfunction.

The term ‘bugs in a computer’ had been used before, but after Grace Hopper wrote in her diary “first actual case of bug being found” the term became really popular, and that’s why we are still using it today.

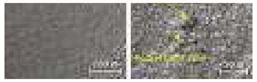
Grace Hopper

Spectacular Software Failures

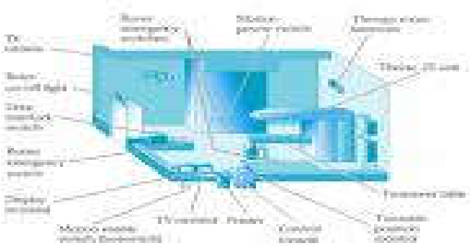
- NASA's Mars lander: September 1999, crashed due to a units integration fault



Mars Polar Lander crash site?



- THERAC-25 radiation machine : Poor testing of safety-critical software can cost *lives* : 3 patients were killed



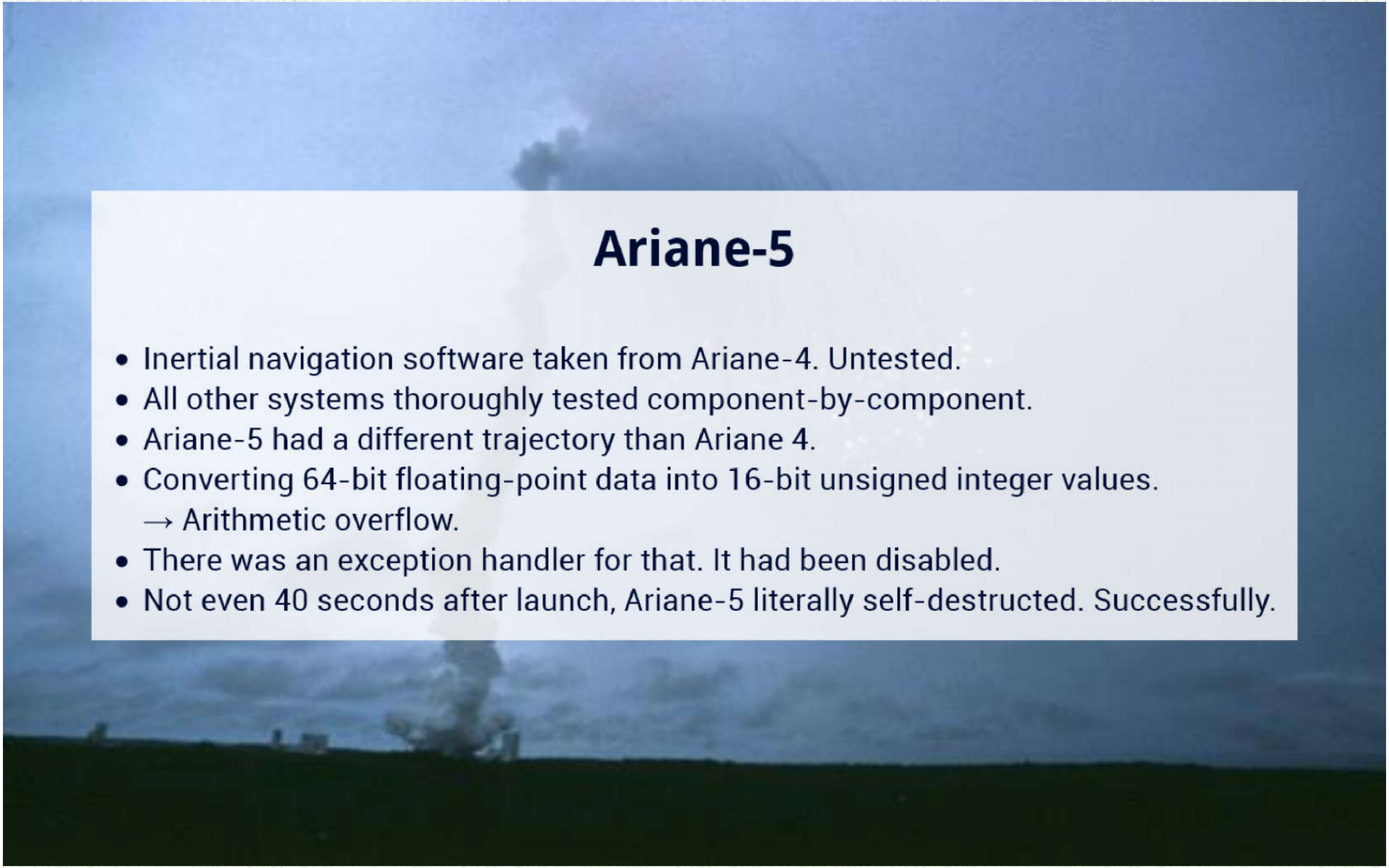
THERAC-25 design

- Intel's Pentium FDIV fault :

Public relations nightmare

We need our software to be dependable

Testing is *one* way to assess dependability



Ariane-5

- Inertial navigation software taken from Ariane-4. Untested.
- All other systems thoroughly tested component-by-component.
- Ariane-5 had a different trajectory than Ariane 4.
- Converting 64-bit floating-point data into 16-bit unsigned integer values.
  → Arithmetic overflow.
- There was an exception handler for that. It had been disabled.
- Not even 40 seconds after launch, Ariane-5 literally self-destructed. Successfully.

Inverted Flight

- A developer of the US Air Force *improved* a program for an unmanned rocket:
  - when crossing the equator, flip the coordinates's leading sign.
- The program therefore needed less memory.
- The rocket also made a 180-degrees roll. So what?
- The program later was used for the autopilot of an F-18 fighter jet.
- When crossing the equator, the pilot certainly was somewhat suprised.

Heartbleed

- Heartbeat Extension of TLS is used to keep Datagram TLS sessions open.
- Simple request and response scheme:
  - *“Send me back the following (padded) string which is* ***n*** *bytes long.„*
- An attacker just had to request a long string, while telling it is short.
- Other party responded with short string, then leaked potentially confidential data.

Northeast Blackout of 2003

508 generating units and 256 power plants shut down

Affected 10 million people in Ontario, Canada

Affected 40 million people in 8 US states

Financial losses of

$6 Billion USD

The alarm system in the energy management system failed due to a software error and operators were not informed of the power overload in the system

Costly Software Failures

- NIST report, “The Economic Impacts of Inadequate Infrastructure for Software Testing” (2002)
  - Inadequate software testing costs the US alone between $22 and $59 billion annually
  - Better approaches could cut this amount in half
- Huge losses due to web application failures
  - Financial services : $6.5 million per hour (just in USA!)
  - Credit card sales applications : $2.4 million per hour (in USA)
- In Dec 2006, *amazon.com’s* BOGO offer turned into a double discount
- 2007 : Symantec says that most security vulnerabilities are due to faulty software

World-wide monetary loss due to poor software is staggering

Validation

The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

—IEEE Std 610.12-1990

“Are we building the right product?„

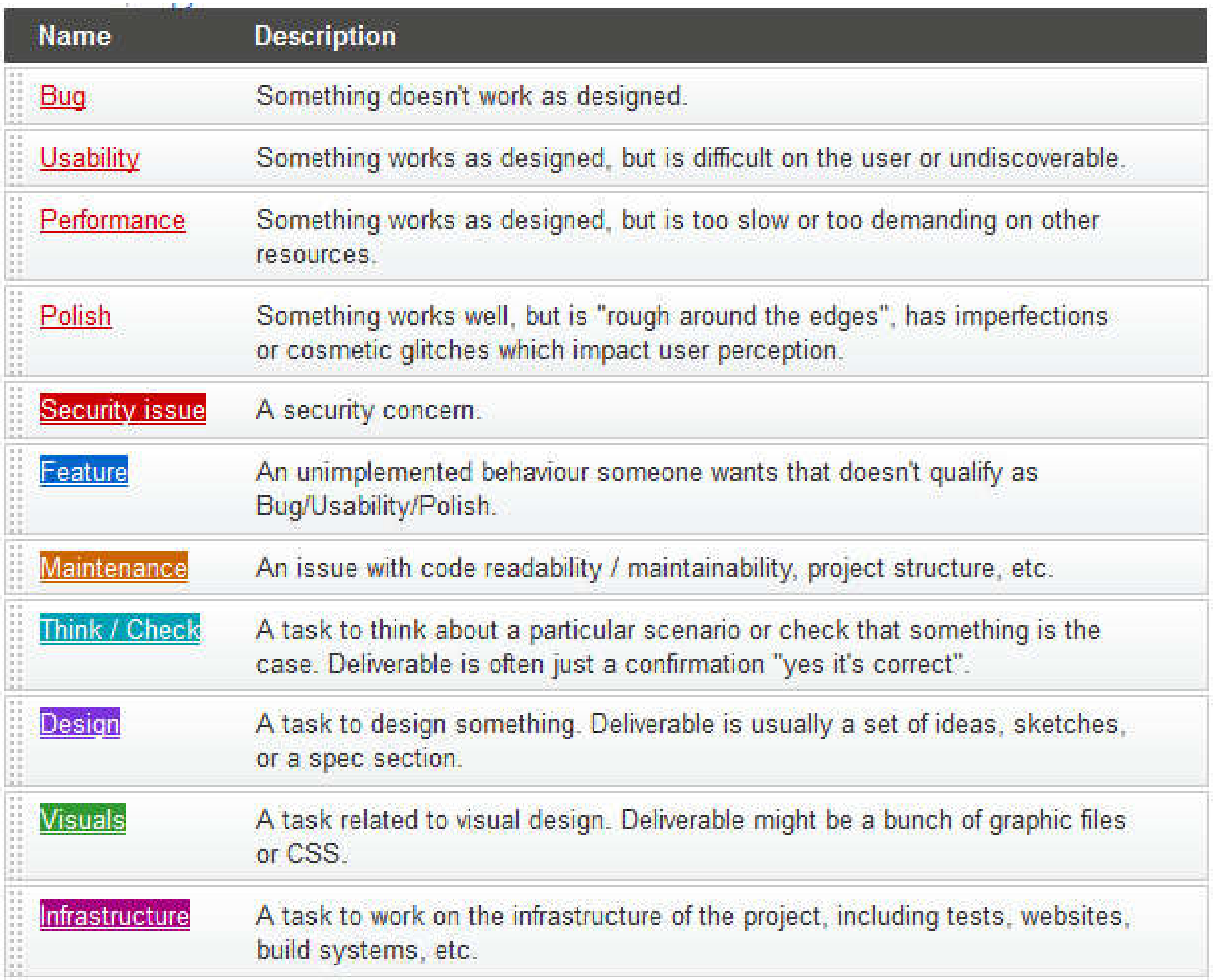
Verification

The process of evaluating a system or component to determine whether the system of a given development phase satisfies the conditions imposed at the start of that phase.

—IEEE Std 610.12-1990

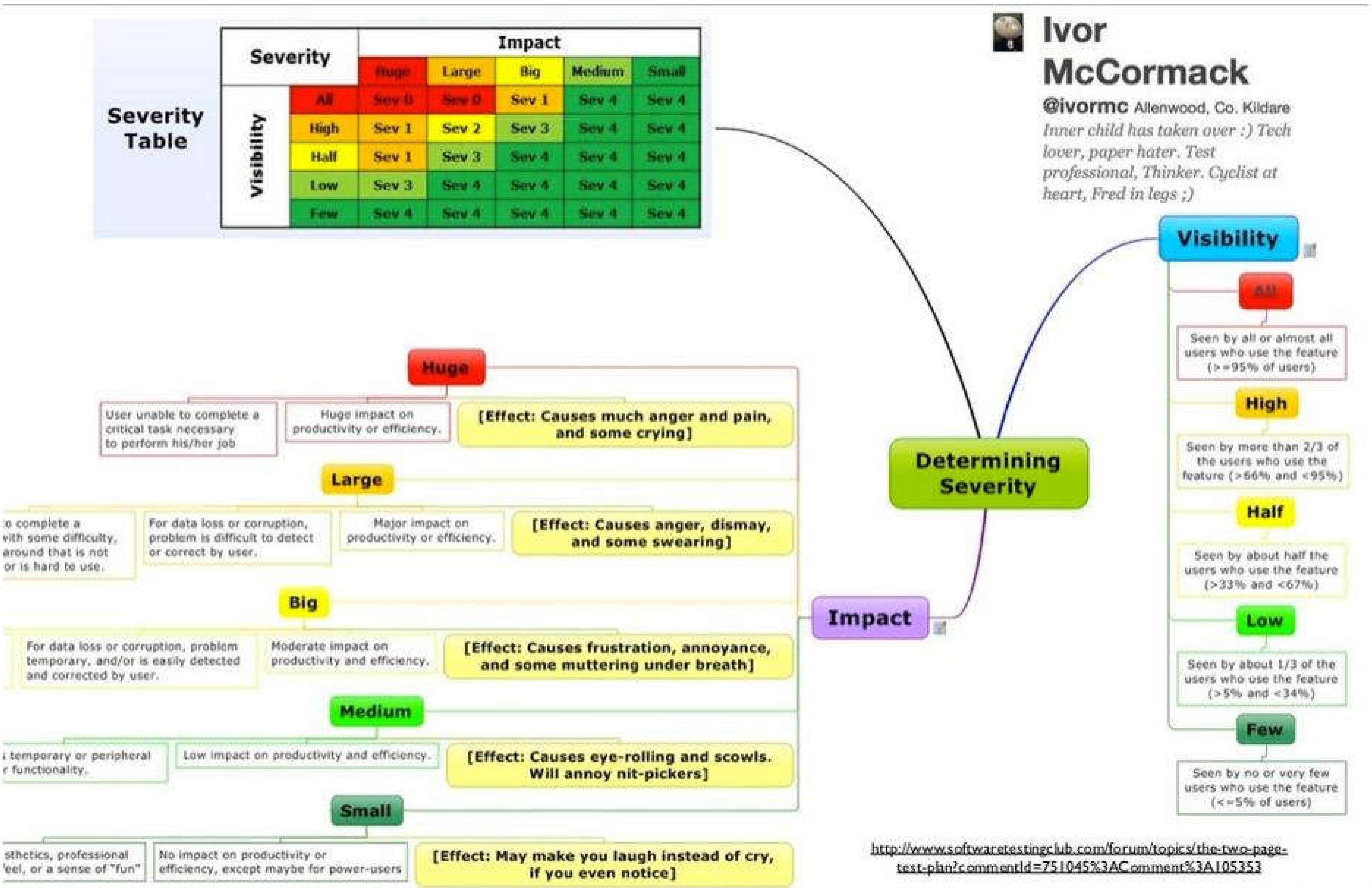
“Are we building the product right?„

Types of bugs in software



Name	Description
Bug	Something doesn't work as designed.
Usability	Something works as designed, but is difficult on the user or undiscoverable.
Performance	Something works as designed, but is too slow or too demanding on other resources.
Polish	Something works well, but is "rough around the edges", has imperfections or cosmetic glitches which impact user perception.
Security issue	A security concern.
Feature	An unimplemented behaviour someone wants that doesn't qualify as Bug/Usability/Polish.
Maintenance	An issue with code readability / maintainability, project structure, etc.
Think / Check	A task to think about a particular scenario or check that something is the case. Deliverable is often just a confirmation "yes it's correct".
Design	A task to design something. Deliverable is usually a set of ideas, sketches, or a spec section.
Visuals	A task related to visual design. Deliverable might be a bunch of graphic files or CSS.
Infrastructure	A task to work on the infrastructure of the project, including tests, websites, build systems, etc.

Severity v.s. Priority



Severity Table

Severity		Impact				
		Huge	Large	Big	Medium	Small
Visibility	All	Sev 0	Sev 0	Sev 1	Sev 4	Sev 4
	High	Sev 1	Sev 2	Sev 3	Sev 4	Sev 4
	Half	Sev 1	Sev 3	Sev 4	Sev 4	Sev 4
	Low	Sev 3	Sev 4	Sev 4	Sev 4	Sev 4
	Few	Sev 4	Sev 4	Sev 4	Sev 4	Sev 4

Ivor McCormack

@ivormc Allenwood, Co. Kildare

Inner child has taken over :) Tech lover, paper hater. Test professional, Thinker. Cyclist at heart, Fred in legs ;)

Determining Severity

Visibility

- **All**: Seen by all or almost all users who use the feature (>=95% of users)
- **High**: Seen by more than 2/3 of the users who use the feature (>66% and <95%)
- **Half**: Seen by about half the users who use the feature (>33% and <67%)
- **Low**: Seen by about 1/3 of the users who use the feature (>5% and <34%)
- **Few**: Seen by no or very few users who use the feature (<=5% of users)

Impact

- **Huge**: User unable to complete a critical task necessary to perform his/her job | Huge impact on productivity or efficiency. | [Effect: Causes much anger and pain, and some crying]
- **Large**: o complete a vith some difficulty, around that is not or is hard to use. | For data loss or corruption, problem is difficult to detect or correct by user. | Major impact on productivity or efficiency. | [Effect: Causes anger, dismay, and some swearing]
- **Big**: For data loss or corruption, problem temporary, and/or is easily detected and corrected by user. | Moderate impact on productivity and efficiency. | [Effect: Causes frustration, annoyance, and some muttering under breath]
- **Medium**: temporary or peripheral r functionality. | Low impact on productivity and efficiency. | [Effect: Causes eye-rolling and scowls. Will annoy nit-pickers]
- **Small**: sthetics, professional eel, or a sense of "fun" | No impact on productivity or efficiency, except maybe for power-users | [Effect: May make you laugh instead of cry, if you even notice]

http://www.softwaretestingclub.com/forum/topics/the-two-page-test-plan?commentId=751045%3AComment%3A105353

Testing Goals Based on Test Process Maturity

- Level 0 : There’s no difference between testing and debugging
- Level 1 : The purpose of testing is to show correctness
- Level 2 : The purpose of testing is to show that the software doesn’t work
- Level 3 : The purpose of testing is not to prove anything specific, but to reduce the risk of using the software
- Level 4 : Testing is a mental discipline that helps all IT professionals develop higher quality software

Level 0 Thinking

- Testing is the same as debugging
- Does not distinguish between incorrect behavior and mistakes in the program
- Does not help develop software that is reliable or safe

Level 1 Thinking

- Purpose is to show correctness
- Correctness is impossible to achieve
- What do we know if no failures?
  - Good software or bad tests?
- Test engineers have no:
  - Strict goal
  - Real stopping rule
  - Formal test technique
  - Test managers are powerless

This is what hardware engineers often expect

Level 2 Thinking

- Purpose is to show failures
- Looking for failures is a negative activity
- Puts testers and developers into an adversarial relationship
- What if there are no failures?

This describes most software companies.

How can we move to a team approach ??

Level 3 Thinking

- Testing can only show the presence of failures
- Whenever we use software, we incur some risk
- Risk may be small and consequences unimportant
- Risk may be great and consequences catastrophic
- Testers and developers cooperate to reduce risk

This describes a few "enlightened" software companies

Level 4 Thinking

- Testing is only one way to increase quality
- Test engineers can become technical leaders of the project
- Primary responsibility to measure and improve software quality
- Their expertise should help the developers

This is the way "traditional" engineering works

Where Are You?

Are you at level 0, 1, or 2 ?

Is your organization at work at level 0, 1, or 2 ?

Or 3?

We hope to teach you to become "change agents" in your workplace …

Advocates for level 4 thinking

Try to mark your self-test

We want you to write a set of test cases—specific sets of data—to properly test a relatively simple program.

Here’s a description of the program:
The program reads three integer values from an input dialog. The three values represent the lengths of the sides of a triangle. The program displays a message that states whether the triangle is scalene, isosceles, or equilateral.

White box and black box

Tactical Goals : Why Each Test ?

If you don't know why you're conducting each test, it won't be very helpful

- Written test objectives and requirements must be documented
- What are your planned coverage levels?
- How much testing is enough?
- Common objective – spend the budget … test until the ship-date …
  - –Sometimes called the "date criterion"

Here! Test This!

If you don't start planning for each test when the functional requirements are formed, you'll never know why you're conducting the test

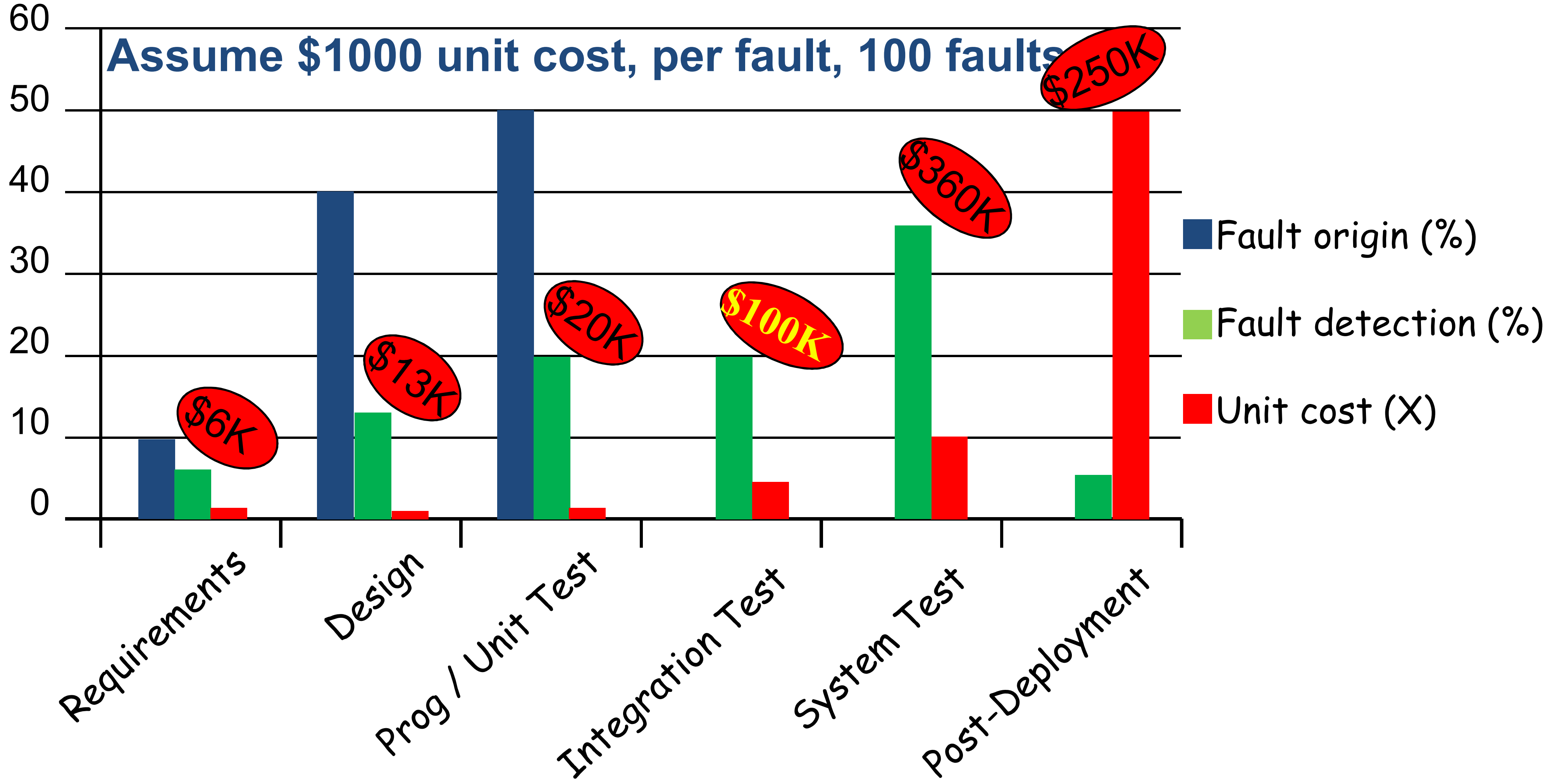
A stack of computer printouts—and no documentation

Cost of Not Testing

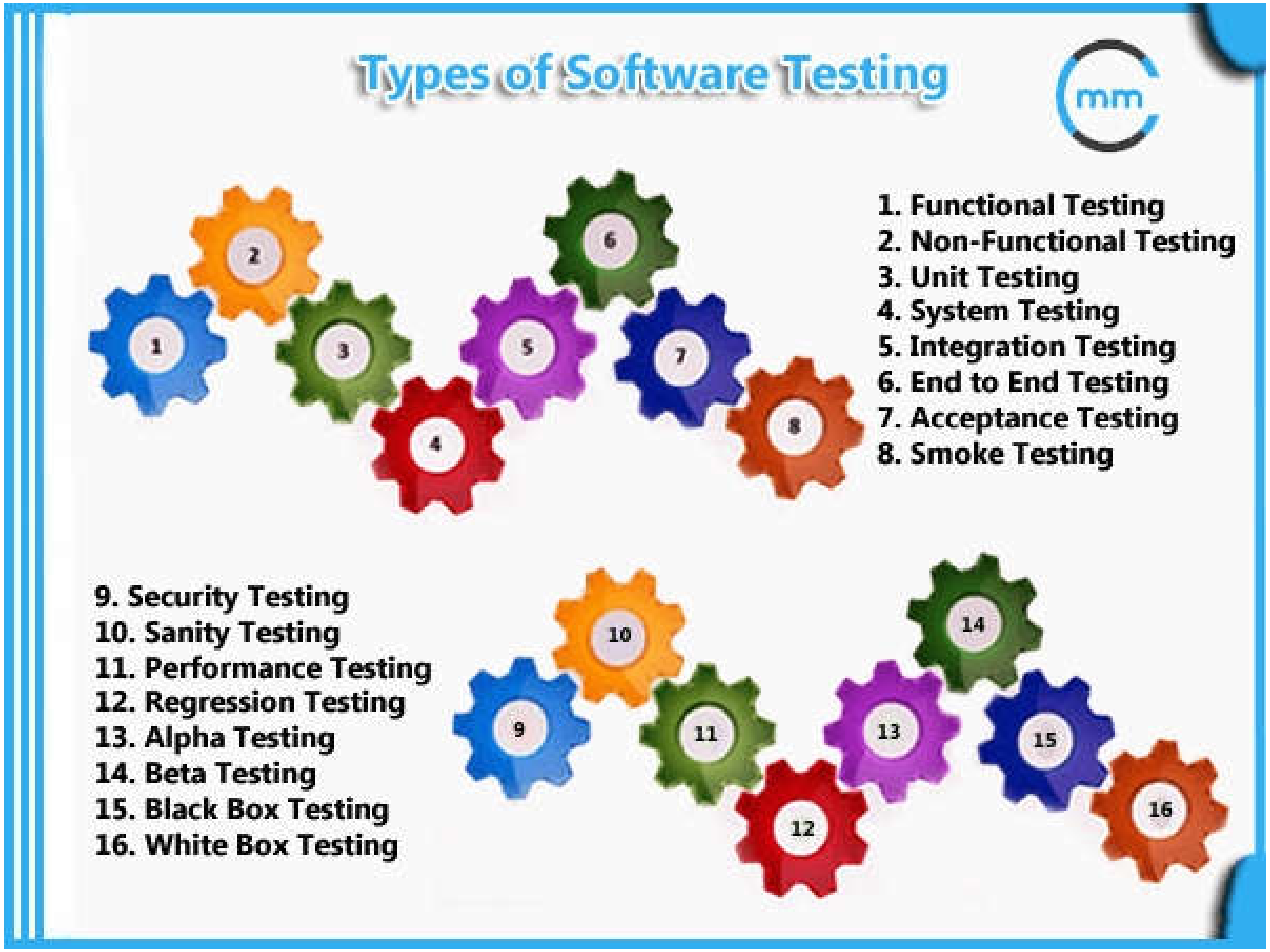
Poor Program Managers might say:
"Testing is too expensive."

- Testing is the most time consuming and expensive part of software development
- Not testing is even more expensive
- If we have too little testing effort early, the cost of testing increases
- Planning for testing after development is prohibitively expensive

Cost of Late Testing



Software Engineering Institute; Carnegie Mellon University; Handbook CMU/SEI-96-HB-002



Types of Software Testing

1. Functional Testing
2. Non-Functional Testing
3. Unit Testing
4. System Testing
5. Integration Testing
6. End to End Testing
7. Acceptance Testing
8. Smoke Testing
9. Security Testing
10. Sanity Testing
11. Performance Testing
12. Regression Testing
13. Alpha Testing
14. Beta Testing
15. Black Box Testing
16. White Box Testing

Static Testing

- Verification only.
- No execution of code.
- Find defects.
- Static analysis (automated).
- Reviews (manual examination, tool support).

Static Analysis

- Analyze code (e.g., control flow, data flow), and generated output.
- Typical use cases for static analysis:
  - syntax checking;
  - code smell detection;
  - coding standards compliance.
- Typical defects found by static analysis:
  - syntax violations;
  - unreachable code;
  - overly complicated constructs.

Reviews

- Different types of reviews:
  - informal review (no formal process; inexpensive);
  - walkthrough (train colleagues and users; gain understanding);
  - technical review (documented, defined process; discuss);
  - inspection (formal process; gain metrics).
- Success factors:
  - clear predefined objectives;
  - defects found are welcomed and expressed objectively;
  - application of suitable review techniques;
  - management supports review process;
  - emphasis on learning and process improvement.

Dynamic Testing

- Primarily verification.
- Execution of code.
- Find failures.
- Different testing methods:
    - black-box testing;
    - white-box testing.
- Different test levels:
    - unit (component, module) testing;
    - integration testing;
    - system testing;
    - acceptance testing.

Black-box Testing

- Specification-based testing.
- Test functionality by observing external behavior.
- No knowledge of internals (required).
- Different black-box testing techniques:
  - equivalence partitioning;
  - boundary value analysis;
  - decision table testing;
  - ...

White-box Testing

- Structure-based testing.
- Close examination of procedural level of detail.
- Knowledge of internals required.
- Different white-box testing techniques:
  - statement testing;
  - decision testing;
  - (multiple) condition testing;
  - ...

Test Levels

Unit Testing

- Test individual units in isolation.
- Find defects in software components (e.g., functions, classes).
- Done by developers.
- In general, white-box tests.

Integration Testing

- Test communication/interaction of units (i.e., their interfaces).
- Maybe separate unit integration and system integration tests.
- Done by developers.
- Primarily white-box tests, but also black-box tests.

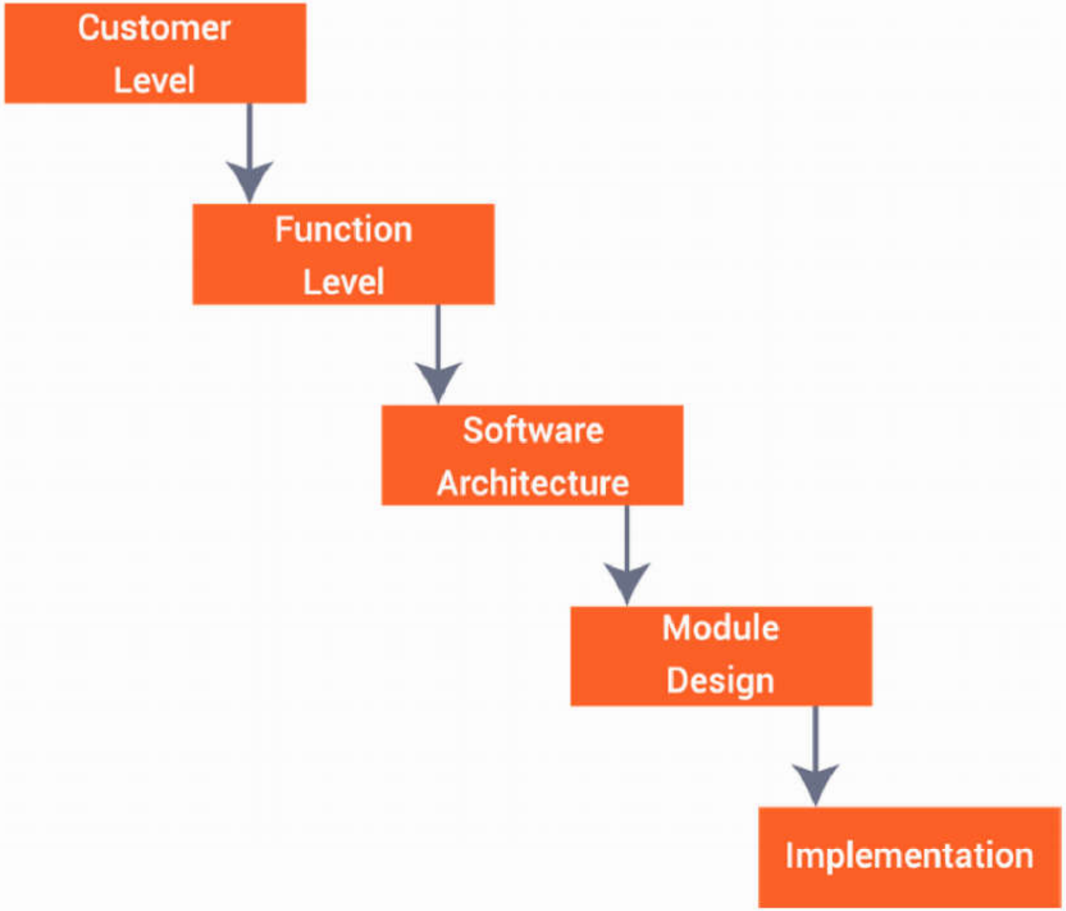
System Testing

- Test complete, integrated system.
- Evaluate system compliance with specified requirements.
- Stress, performance, usability etc. testing.
- Done by (external) testers.
- In general, black-box tests. Additional white-box tests possible.

Acceptance Testing

- Test complete, integrated system.
- Evaluate system compliance with specified acceptance criteria.
- May be performed at various times during development.
- Done by customers/users.
- Only black-box tests.

Development Phases



Customer Level

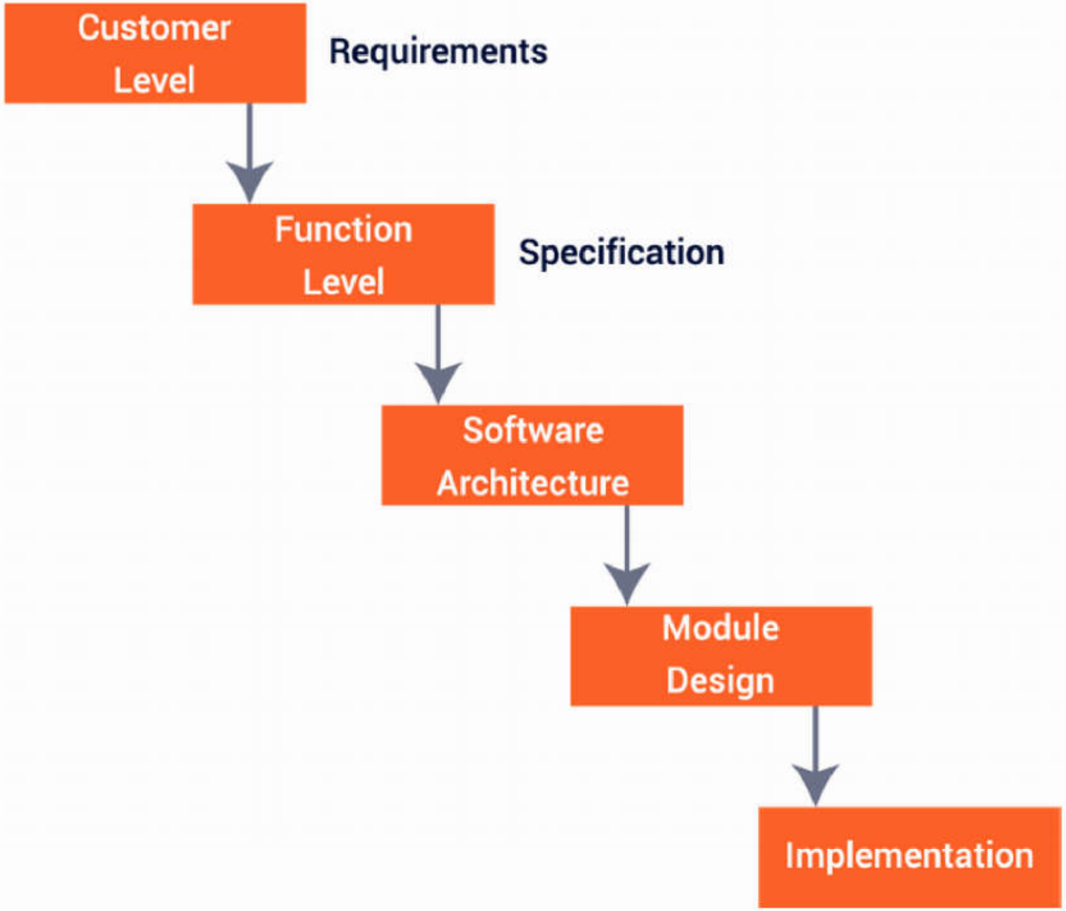
Function Level

Software Architecture

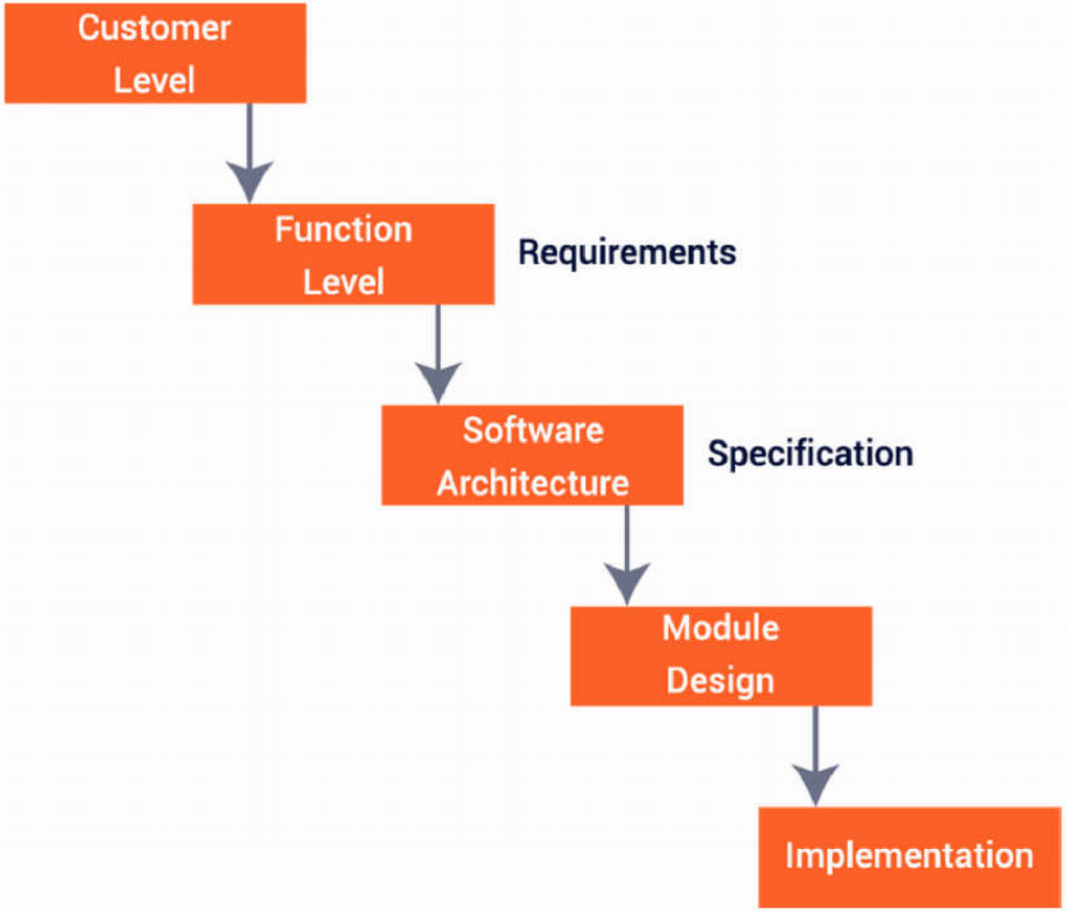
Module Design

Implementation

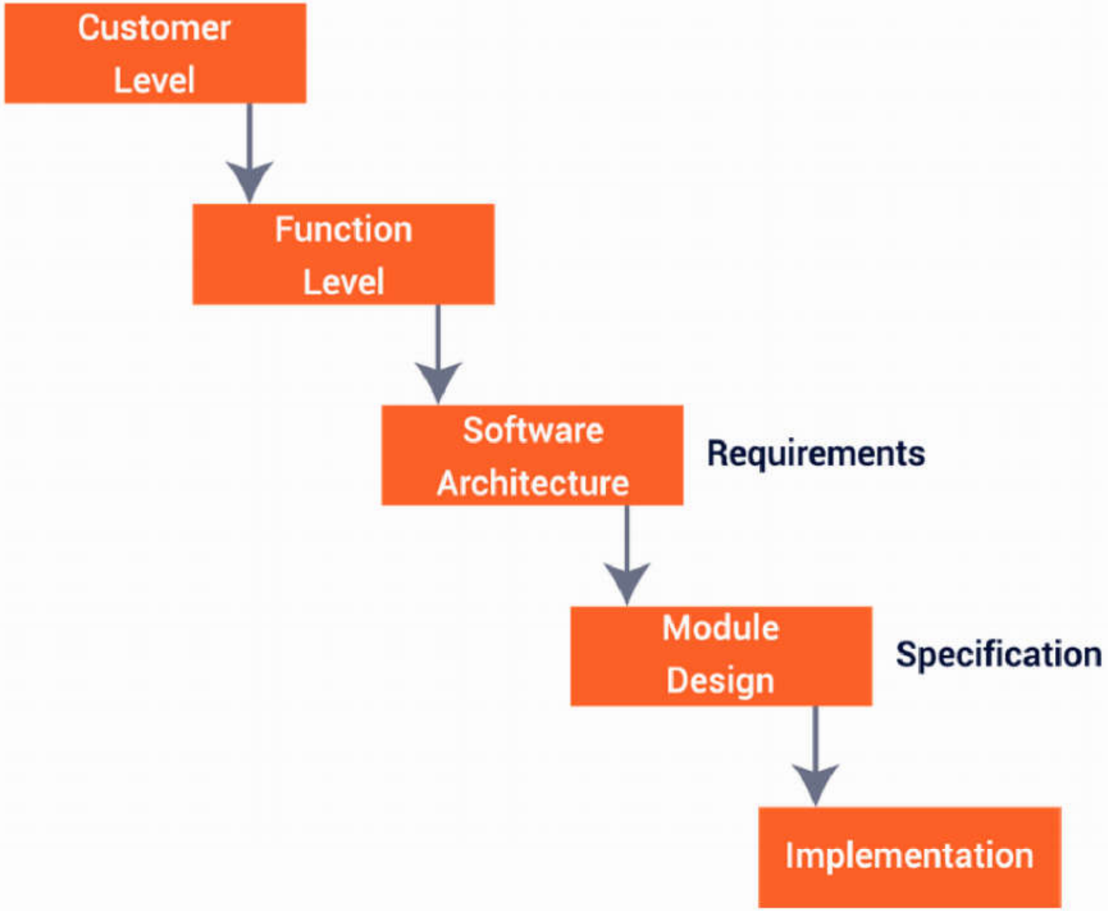
Requirements and Specification



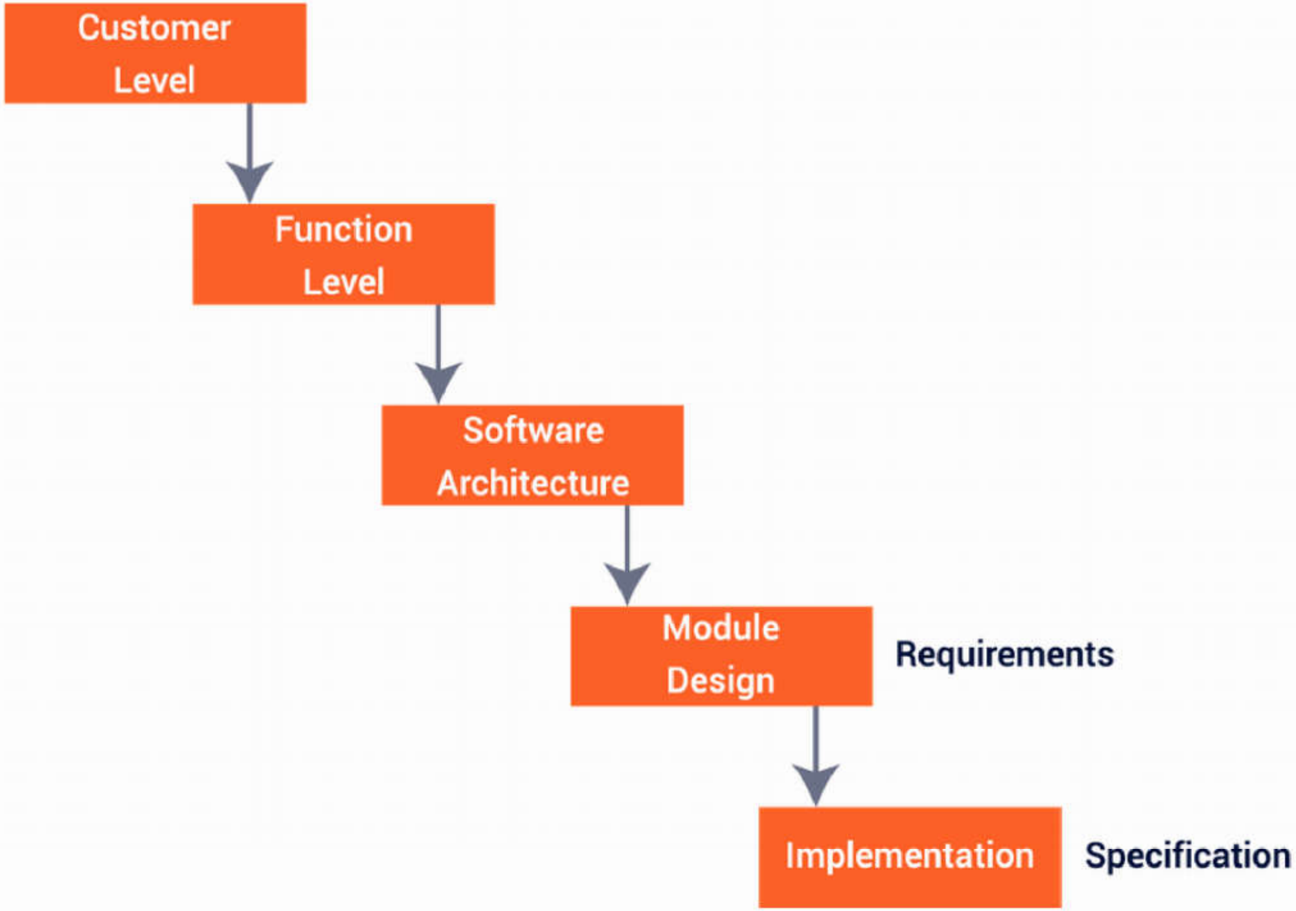
Requirements and Specification



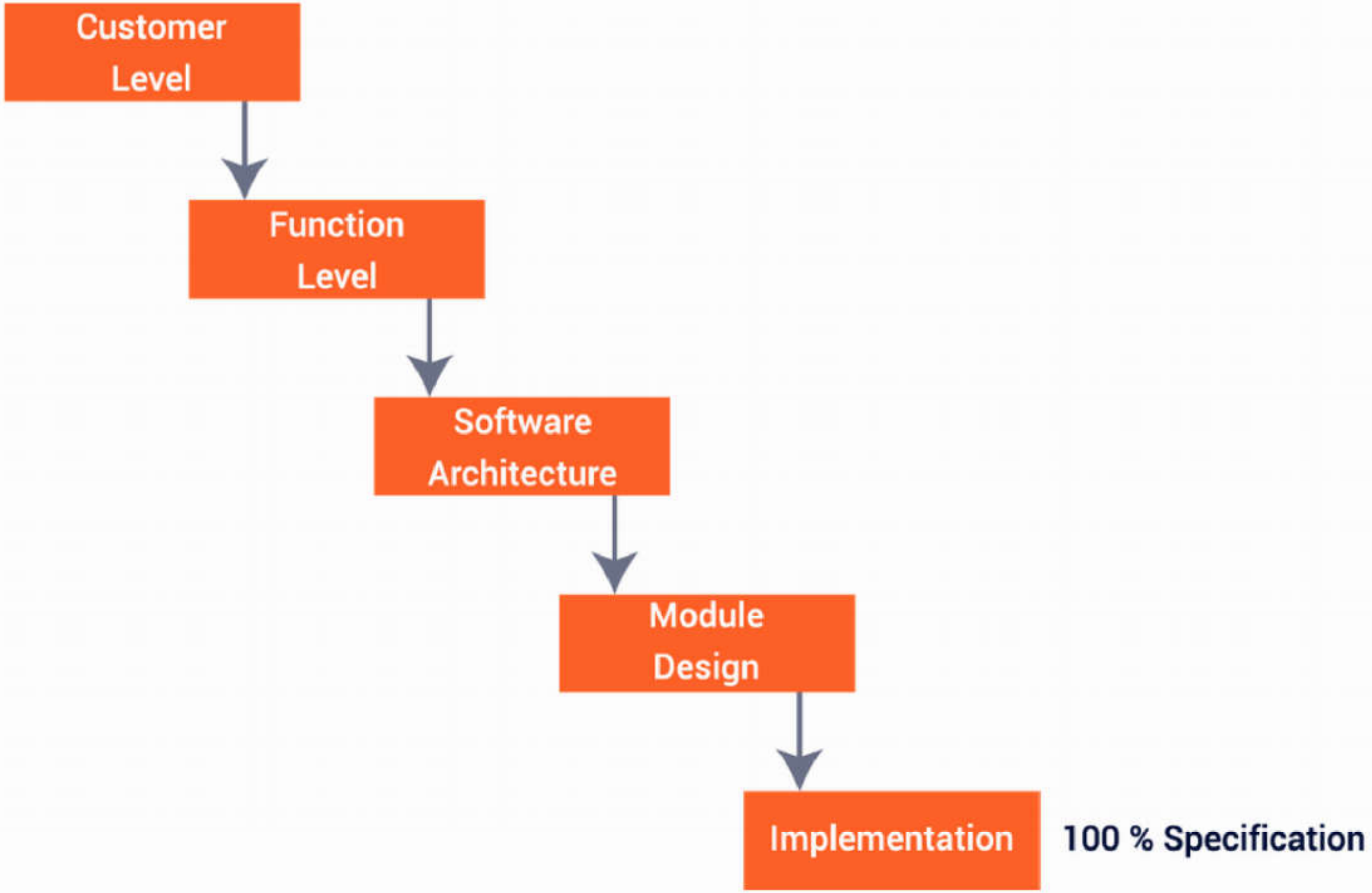
Requirements and Specification



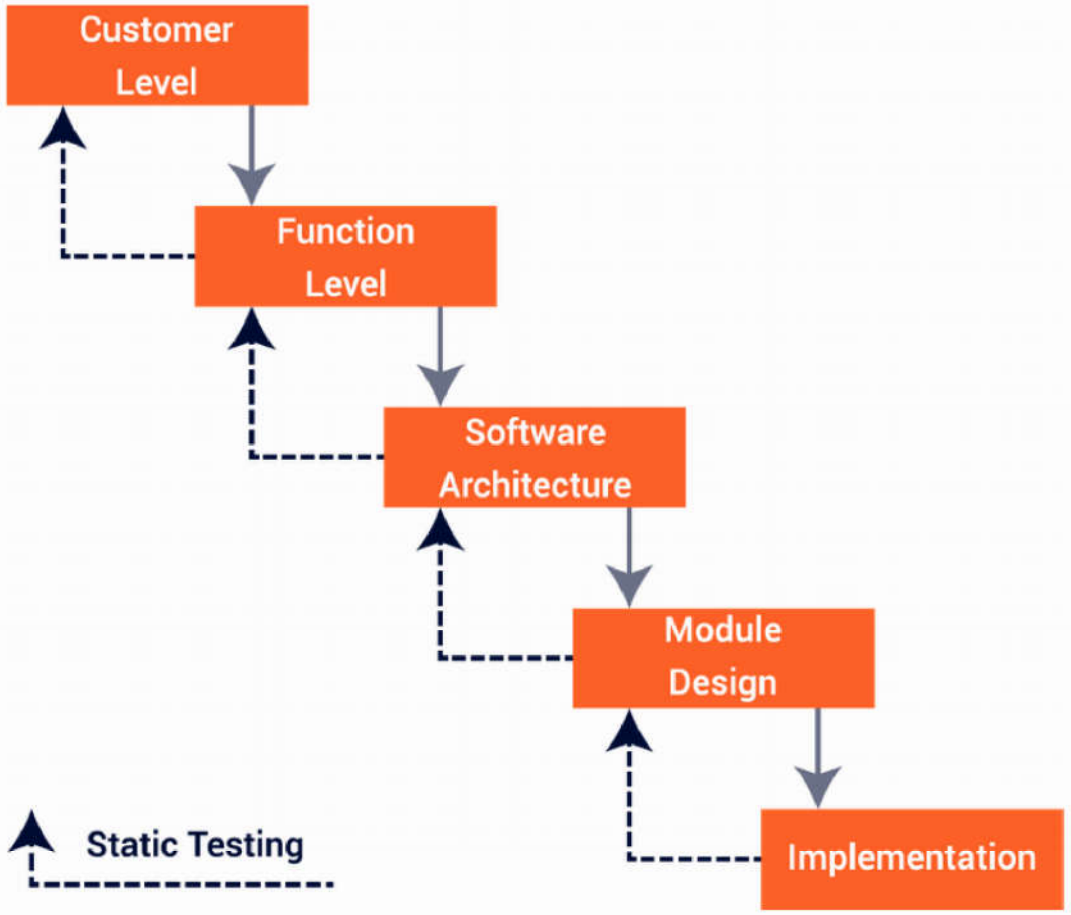
Requirements and Specification



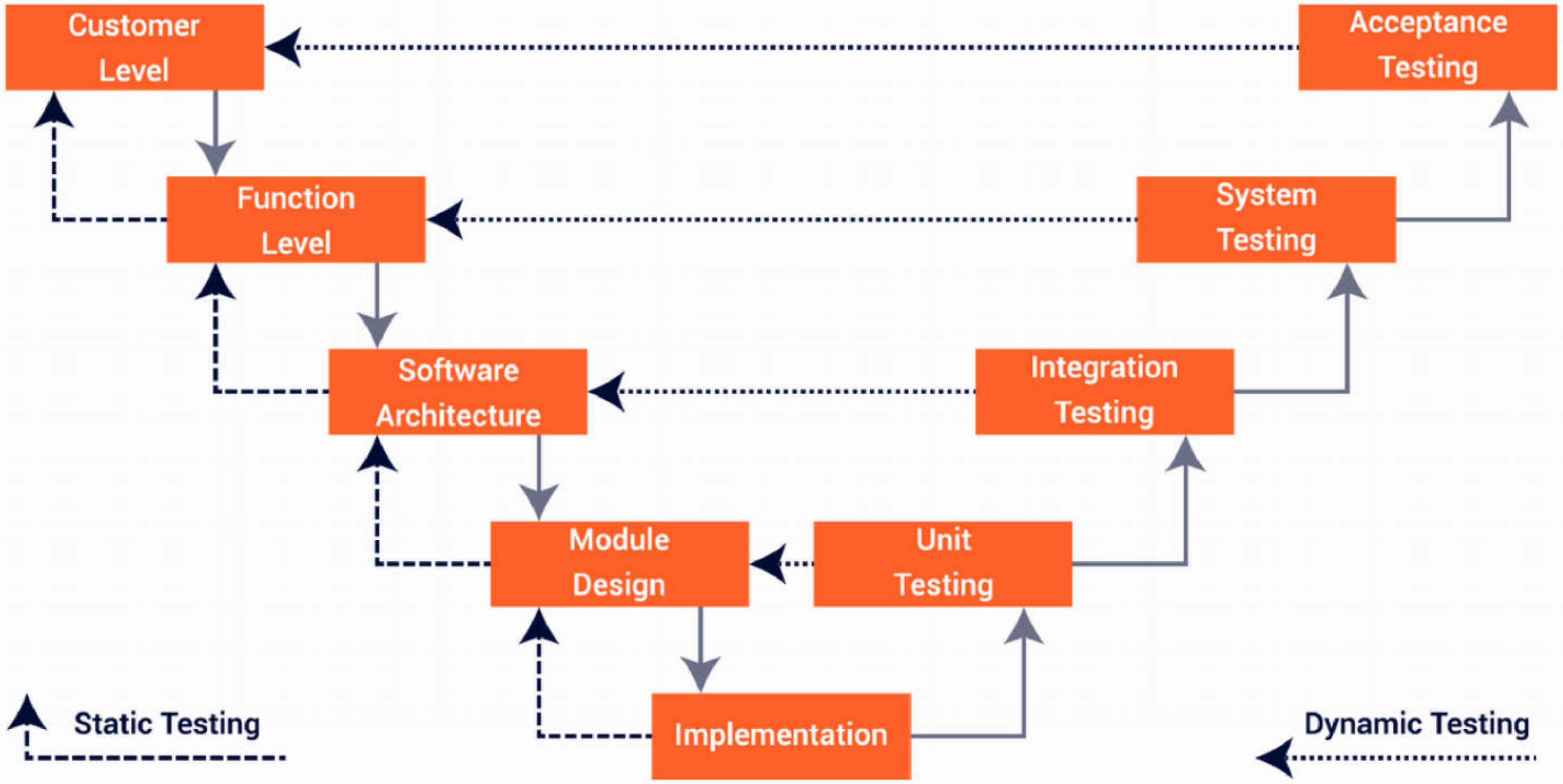
100 % Specification



Static Testing



Dynamic Testing



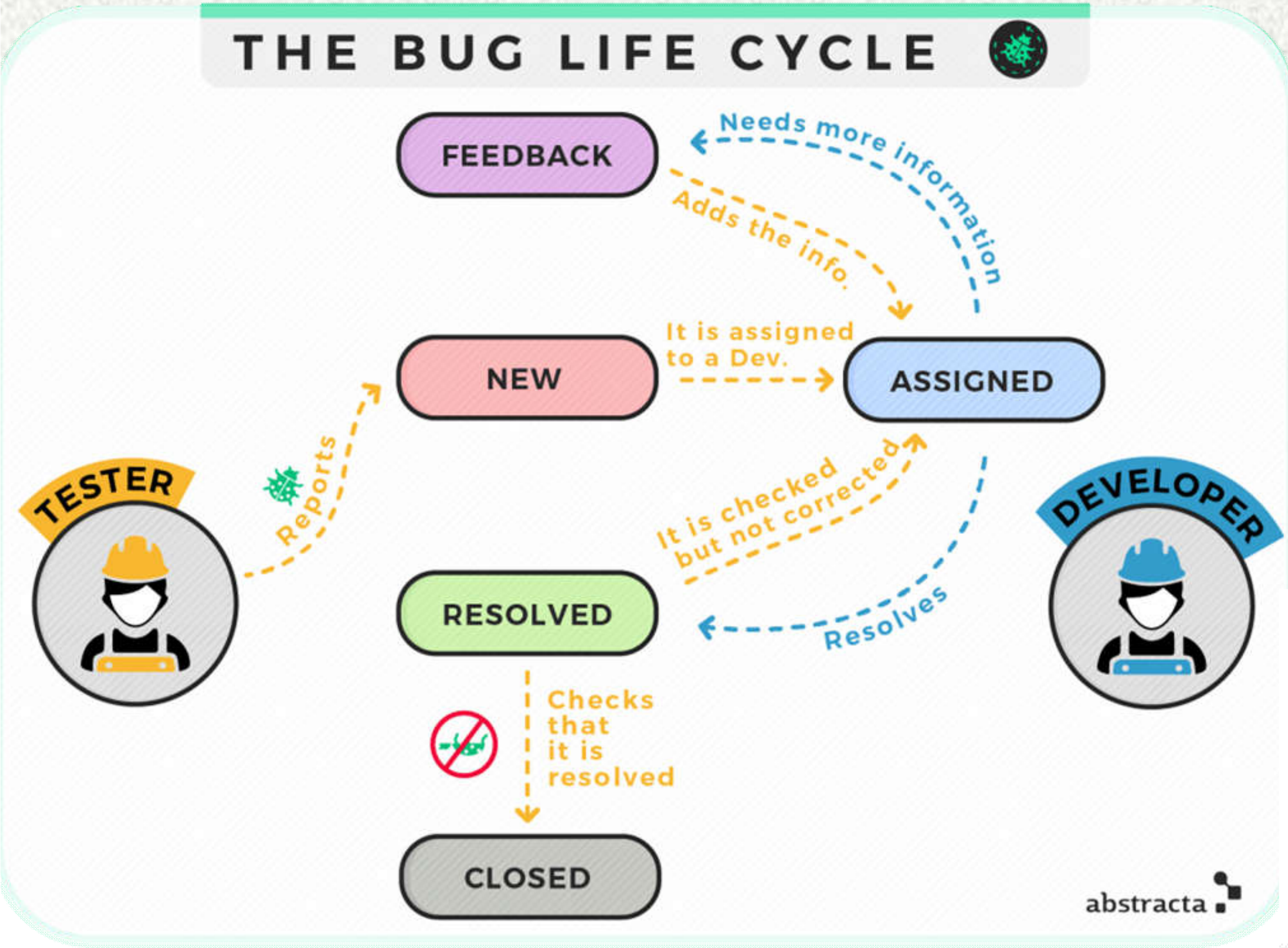
Continuous Integration

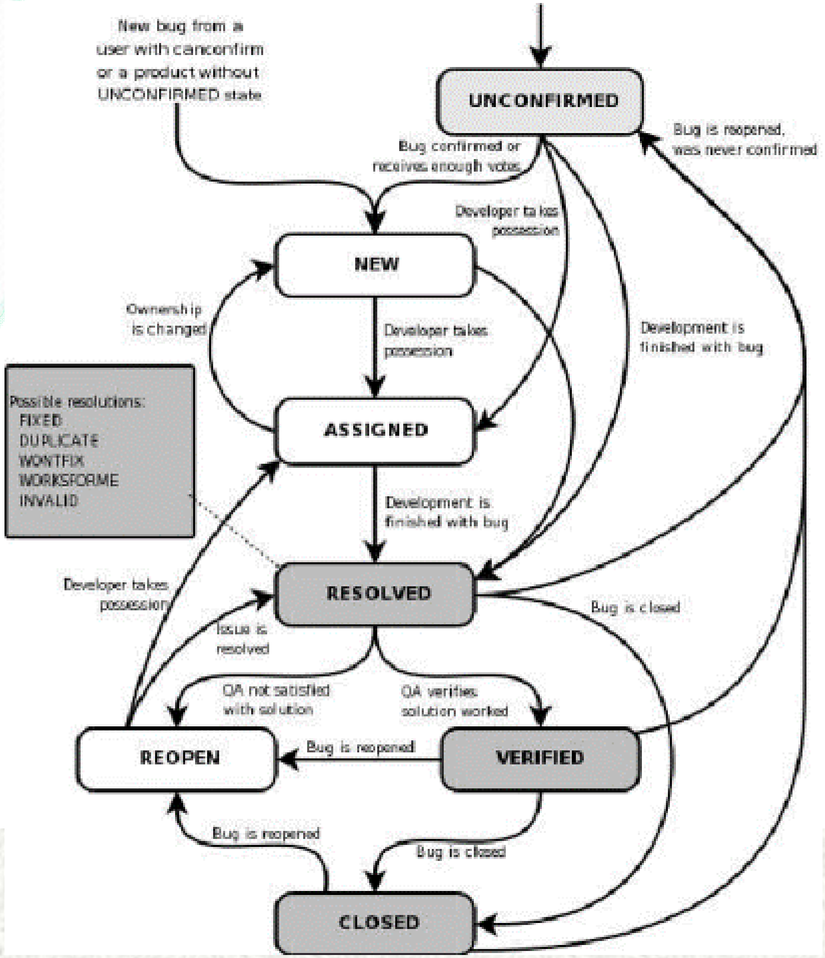
- Automated activities:
  - execute static code analysis;
  - execute unit tests, and check code coverage;
  - deploy to test environment;
  - execute integration tests;
  - ...
- Benefits:
  - earlier detection and analysis of conflicting changes;
  - regular feedback on whether the code is working;
  - no big-bang integration;
  - reduces repetitive manual testing activities;
  - ...

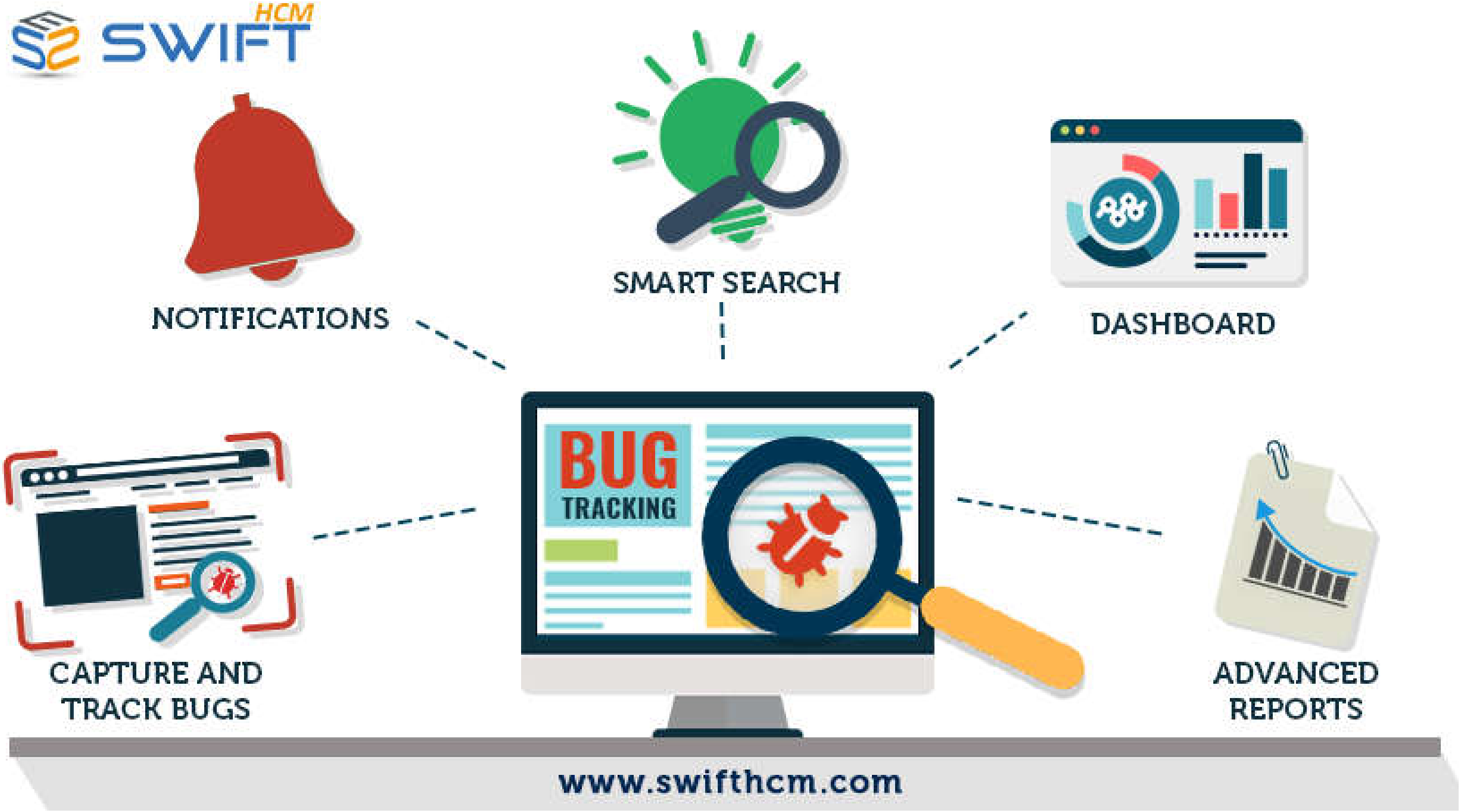
Travis CI

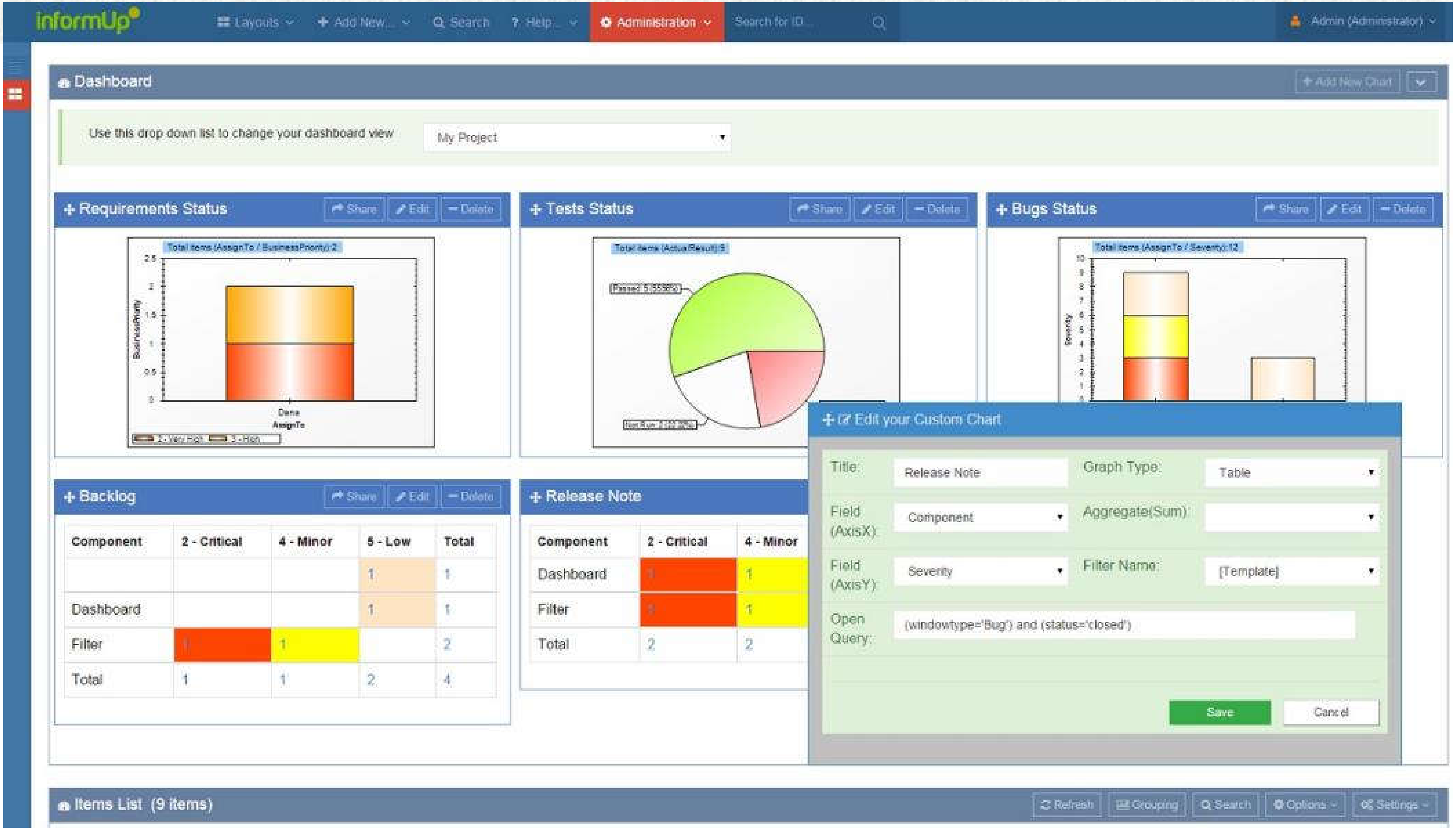
- Hosted continuous integration server.
- Single requirement: GitHub account.
- Getting started:
  - sign in with your GitHub account;
  - accept GitHub access permissions confirmation;
  - set up `.travis.yml` file for your repositories;
  - enable Travis CI builds for individual repositories;
  - develop continuously integrated software...

/** @see travis-ci.org */


THE BUG LIFE CYCLE
FEEDBACK
Needs more information
Adds the info.
NEW
It is assigned to a Dev.
ASSIGNED
TESTER
Reports
It is checked but not corrected
DEVELOPER
RESOLVED
Resolves
Checks that it is resolved
CLOSED
abstracta


New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug confirmed or receives enough votes
Bug is reopened, was never confirmed
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
ASSIGNED
Possible resolutions:
FIXED
DUPLICATE
WONTFIX
WORKSFORME
INVALID
Development is finished with bug
Developer takes possession
RESOLVED
Bug is closed
Issue is resolved
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED


HCM
SWIFT
SMART SEARCH
NOTIFICATIONS
DASHBOARD
BUG
TRACKING
CAPTURE AND
TRACK BUGS
ADVANCED
REPORTS
www.swifthcm.com



Use this drop down list to change your dashboard view: My Project

Backlog

Component	2 - Critical	4 - Minor	5 - Low	Total
			1	1
Dashboard			1	1
Filter	1	1		2
Total	1	1	2	4

Release Note

Component	2 - Critical	4 - Minor
Dashboard	1	1
Filter	1	1
Total	2	2

Edit your Custom Chart

Title:	Release Note	Graph Type:	Table
Field (AxisX):	Component	Aggregate(Sum):	
Field (AxisY):	Severity	Filter Name:	[Template]
Open Query:	(windowtype='Bug') and (status='closed')		

Reliability

- G. Myers:
- “Software reliability is the probability that the software will execute for a particular period of time without a failure, weighted by the cost to the user of each failure encountered.”
- “reliability as a quality measure of the lack of errors in a program”

Approaches towards reliability

- **Fault avoidance**
  - Minimizing or eliminating errors
- **Fault detection**
  - Software functions that reveal errors
- **Fault correction**
  - Software functions that correct errors or their damage
- **Fault tolerance**
  - Ability to continue operation in the presence of error

Fault avoidance

- **The optimal approach towards software reliability**
- **Techniques**
  - Minimizing complexity
  - Improving the communication
  - Improving the translation
  - Detecting errors at each translation step

Fault detection

- To detect as many faults as possible
- To detect faults as early as possible

Fault correction

- **Possible for the hardware**
  - replace the faulty part with the spare part
- **Very limited progress in software**
  - Same fault exists in the duplicate copies of the software
  - Damage correction technique
    - Predict damage in advance (how much we can predict …)
    - Design software functions to undo the damage (how much we can undo…)

Fault tolerance

- **Error isolation**
  - Isolated functions become inoperable
- **Fallback**
  - Commonly used in operating systems
- **Dynamic redundancy**
  - N-version programming
    - Different designs, algorithms, programming languages, etc to implement the same function
    - Question: which versions return the correct result

Some Guidelines for Debugging

- Analyze and think carefully
  - Avoid experimentation
- Not just fix one instance of the error
  - Errors tend to cluster
- Correction may give rise to new errors
  - Explain the problems to others

Error Analysis

- How was the error found
  - What
  - Where
  - When
- What was done incorrectly
- Why was not detected earlier