# Software Testing

A general introduction

# Assessment

- 10% attendance (2.5’ each)
- 40% course project
  - Group of 4
  - Report 30’
  - Presentation 10’
- 50% final examination (Date and location to be announced)

# Attendance

- Sign your name on the attendance list before class begins or during the first breaks.
- 2.5 points each attendance
- DO NOT sign for others

# Course Project: Report submission

- Maximum 4 pages (exclusive of appendix and references), in ACM format, which is available at https://www.acm.org/publications/proceedings-template
- Written in English, submit pdf file
- Contents (format)
  - 1. What problem does the paper solve?
  - 2. Why is the problem important?
  - 3. How is the problem being solved?
  - 4. Compare to existing approaches, what is the improvement of the approach?
  - 5. How is the approach evaluated? (Including dataset/benchmark, experiment design and results)
  - 6. How can the proposed approach be used in real world scenarios?
  - 7. What are the potential future work?
- Deadline: June 4th (Monday), 11:59pm
- Email to shuang.liu@tju.edu.cn, (cc all group members)
  Email title: **[Group #] Software Testing Report**

# Course Project: Final presentation

- Time: May 28th , 1:30pm – 5:00pm,
- Location: 31-259
- 10 min each team (schedule will be available near the presentation)
  - 8min presentation + 2min Q&A
- Contents: Follow the contents of your report
- Only 1 person present your findings, all the other team members should also attend to answer questions.
- We are serious about timing, so be prepared and do rehearse.

# Marking Criteria—Report

- All required formats are followed 5′
- All required contents are presented 5′
- The written English is formal, no typos, no gramma errors. 5′
- Show correct understanding of the paper by concisely summarize "What, Why and How" 5′
- A comprehensive summary and comparison with existing work, based on experiment results where necessary. 5′
- Properly proposed future direction (maybe inspired by real world scenarios), with theoretical and practical support 5′
- Large paragraphs of copying from the paper -10′

# Marking Criteria—Presentation

- All required contents are presented 3′
- Speak clearly, confidently, loudly enough to be heard 5′
- Answer questions correctly and politely 2′

# TODO…

- Form group of 4 on your own
- You will finish your course project in groups, and obtain group marks
- Deadline of forming groups: 4pm, Wednesday, May 9th
- Each group send an email to shuang.liu@tju.edu.cn (cc all group members) to inform me about your group members. Title for the email would be "**Software Testing Group Allocation**", body of the email list all the members' name and metric number.
- Contact me before May 11th if you cannot find a group.

# References

- Text books → No compulsory text books
- Reference
  - Any basic software testing book
  - Papers of course projects
  - Good technical websites for software testing
  - http://softwaretestingfundamentals.com
  - Wiki pages

# Module Outline

- Introduction to software testing
- Review Classical testing methods
- New trends and topics in software testing
- Course Project Presentation

# Introduction to Software Testing--Outline

- Software Quality
- Software Testing
  - Testing Levels
  - Testing Types
  - Testing Methods
  - Coverage Criteria
  - Testing Activities

# Software Quality

# Software Quality


- Based on the Consortium for IT Software Quality (CISQ), the followings need to be achieved
  - **Reliability**: An attribute of resiliency and structural solidity. Reliability measures the level of risk and the likelihood of potential application failures. It also measures the defects injected due to modifications made to the software (its "stability" as termed by ISO).
  - **Efficiency**: The source code and software architecture attributes are the elements that ensure high performance once the application is in run-time mode.
  - **Security**: A measure of the likelihood of potential security breaches due to poor coding practices and architecture. This quantifies the risk of encountering critical vulnerabilities that damage the business.
  - **Maintainability**: Maintainability includes the notion of adaptability, portability and transferability (from one development team to another).
  - **Size**: While not a quality attribute per se, the sizing of source code is a software characteristic that obviously impacts maintainability.

# Real world scenarios—Functional I
## Therac-25 (radiation therapy machine)


- When: June 1985 -January 1987
  https://hackaday.com/2015/10/26/killed-by-a-machine-the-therac-25/
- What: 4 patients were killed and 2 patients were left with lifelong injuries
- Why: due to massive overdoses of radiation
- Software error: arithmetic overflow. The software set a flag variable by incrementing it. Occasionally an arithmetic overflow occurred.
- Lessons learnt: Bad software design and development practice
  - Code was NOT independently reviewed
  - Never tested with the combination of software and hardware until it was assembled at the hospital

# Real world scenarios—Functional II
## 2003 North America Blackout

- When:  2003
- What: Power Outage throughout parts of the Northeastern and Midwestern United States and the Canadian province of Ontario
- Estimated 55 million people were affected
- Why:  a software error in the alarm system at a control room of the FirstEnergy Corporation, located in Ohio

# Real world scenarios—Functional III
## Ariane 5 Rocket Crash

- When: June 4, 1996
- What:  Ariane 5 Rocket broke up and exploded 40 seconds after launch

  Cost: 10 years hard work (European Space Agency) and $7 billion
- why: due to an arithmetic overflow
  - The 64-bit floating point number had a value too large to be represented by a 16-bit signed integer

http://www-users.math.umn.edu/~arnold/disasters/ariane5rep.html

# Real world scenario – Performance I

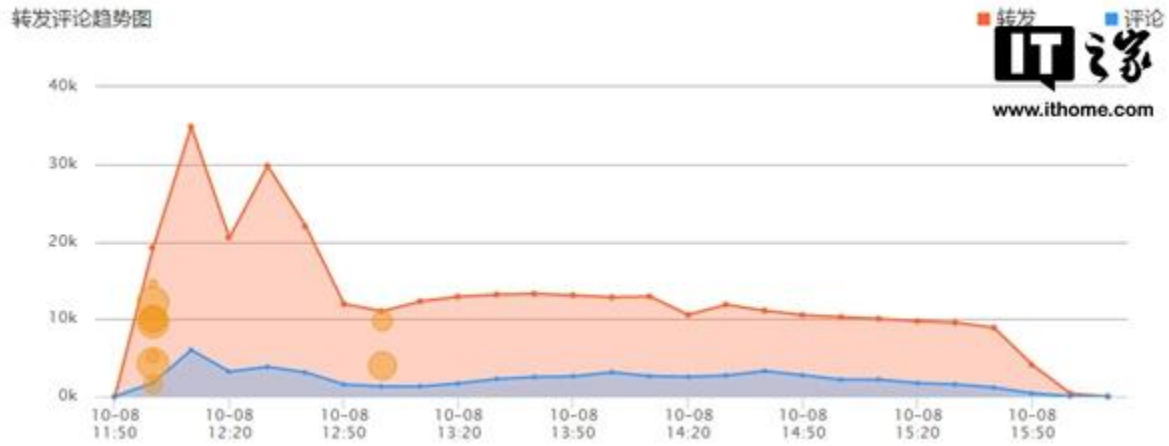
EVENT ANNOUNCEMENT

**Due to high demand for the following event(s), we are regulating the traffic so that you can complete your purchase smoothly.**

Please also ensure that you're using one of the following browsers when purchasing tickets online:

Internet Explorer 10+, Firefox 27+, Google Chrome 34+ or Safari 7+.


JACKY CHEUNG "A CLASSIC TOUR"

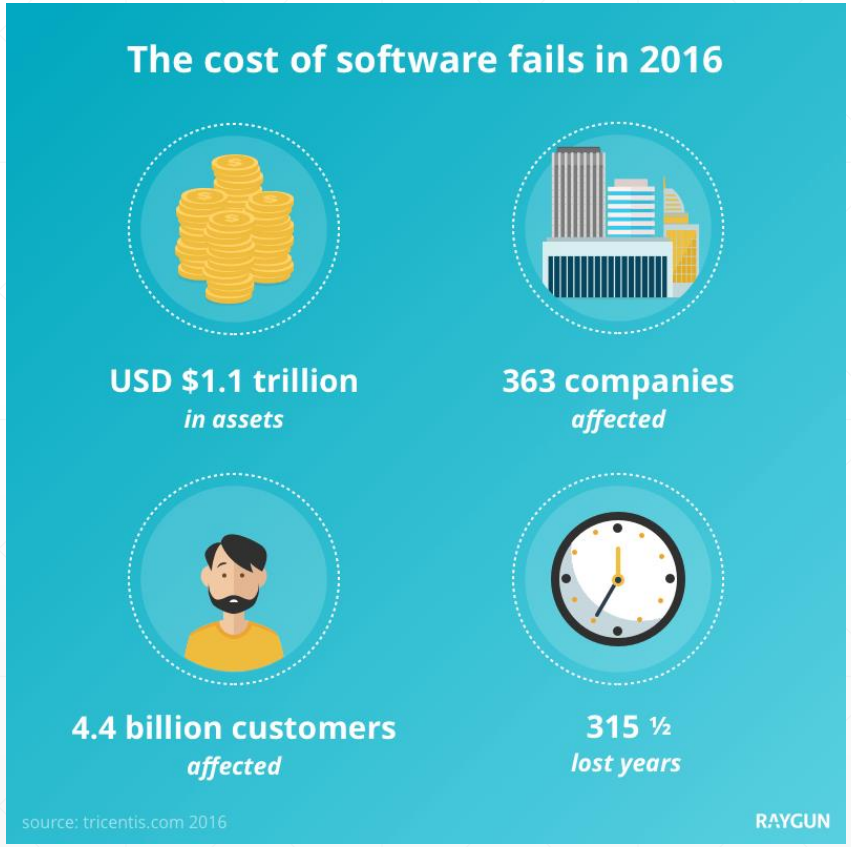



# Real world scenario – Performance II
## 2015. 11. 11 Statistics (Tmall)

- 18 seconds, ￥ 100 million
- 72 seconds, ￥ 1 billion
- 5 min 45 seconds, ￥ 5 billion
- 12 min 28 seconds, ￥ 10 billion
- …
- End of day, ￥ 91.217 billion
- 120,000 transactions/min
- First package delivered 14 mins after starts…
- In total 105.8 million transactions, more than 200 million packages…

# Cost of Software Errors

- An estimated $59.5 billion annually (for United States)
- About 0.6 percent of the gross domestic product (GDP)
- More than a third of these costs (an estimated $22.2 billion) could be eliminated by an improved testing infrastructure, even though not all errors can be removed

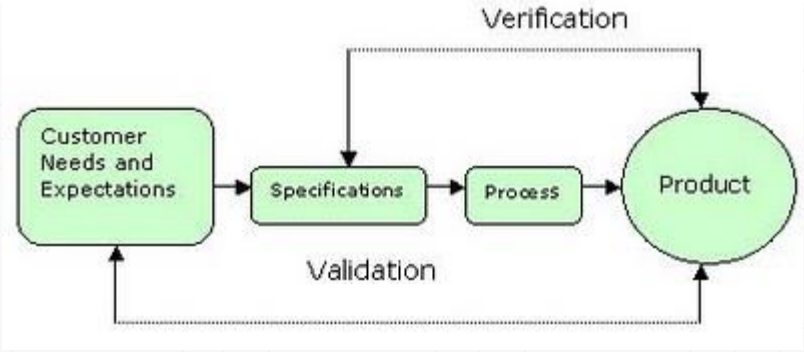


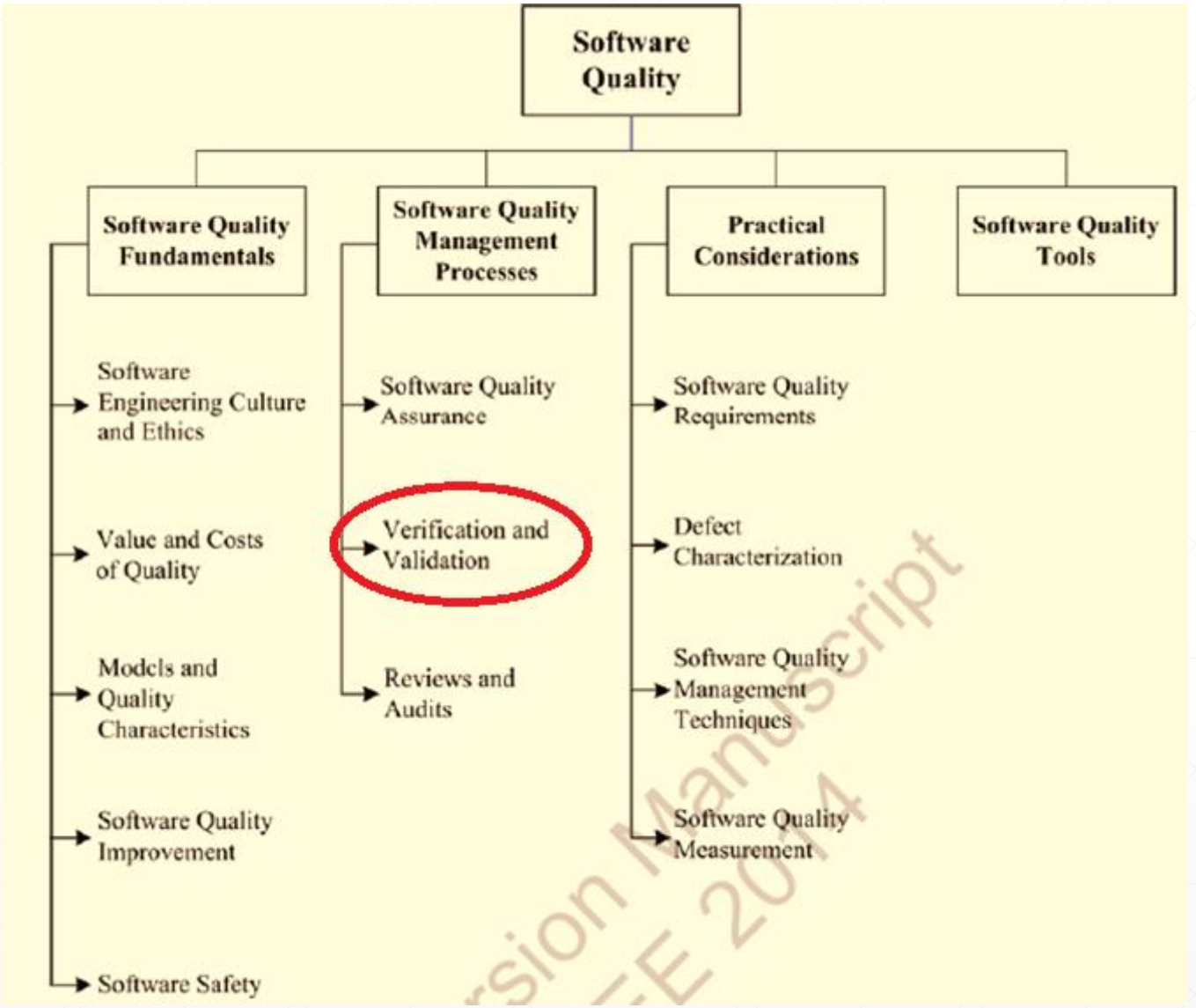

https://raygun.com/blog/cost-of-software-errors/

# 5 min break


# V&V

- **Verification:** Have we built the software right? (i.e., does it match the specification).
- **Validation:** Have we built the right software? (i.e., is this what the customer wants).



# Verification vs. Validation

| Verification | Validation |
| --- | --- |
| To check whether the software conforms to specifications. | To check whether software meets the customer expectations and requirements. |
| Target is requirements specification, application and software architecture, high level, complete design, and database design etc. | Target is actual product-a unit, a module, a bunch of integrated modules, and effective final product. |
| It can catch errors that validation cannot catch. It is low level exercise. | It can catch errors that verification cannot catch. It is High Level Exercise. |
| It generally comes first, done before validation. | It generally follows after verification. |

# Why V & V

## Goals:

- Establish confidence that software is fit for its intended purpose
  - This does NOT mean completely free of defects
  - Rather, it must be good enough for its intended use and the type of use will determine the degree of confidence that is needed

## Confidence Parameters:

- Software function
  - How critical is the software to the organization?
- User expectation
  - Users may have low expectations of certain kinds of software.
- Marketing environment
  - Getting a product to market early might be more important than finding all defects

# Common types of Software faults I

- Arithmetic faults
  - Division by zero.
  - Arithmetic overflow.
  - Loss of arithmetic precision due to rounding or numerically unstable algorithms.
- Logic faults
  - Infinite loops and infinite recursion.
  - Off-by-one error, counting one too many or too few when looping.
- Syntax faults
  - Use of the wrong operator, such as performing assignment instead of equality test. For example, in some languages x=5 will set the value of x to 5 while x==5 will check whether x is currently 5 or some other number.

# Common types of Software faults II

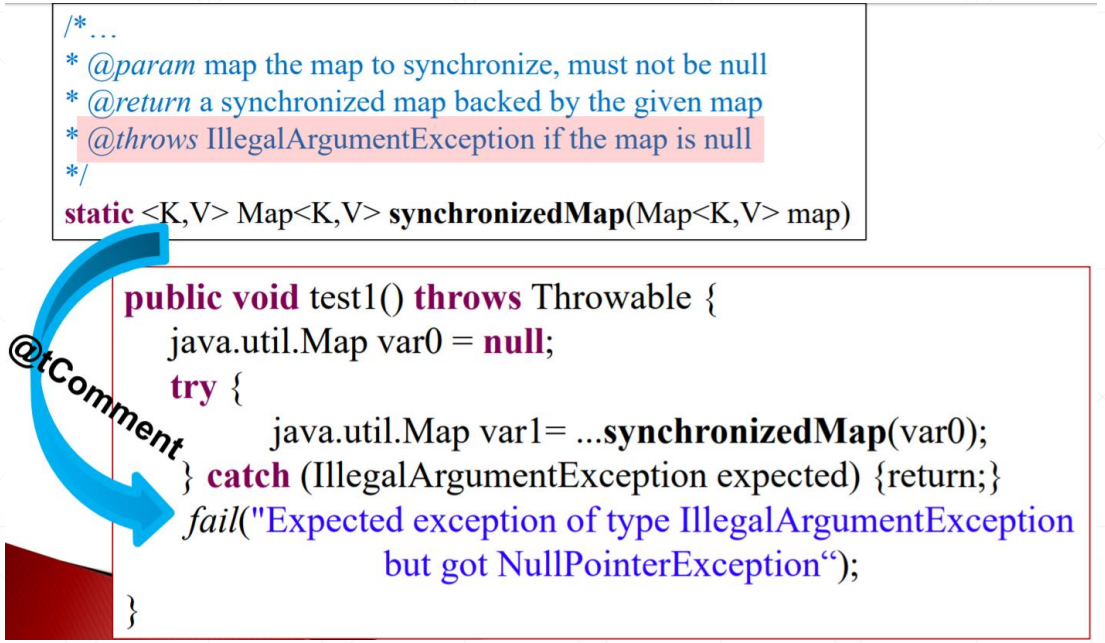
- Resource faults
  - Null pointer dereference.
  - Using an uninitialized variable.
  - Access violations.
  - Resource leaks, where a finite system resource (such as memory or file handles) become exhausted by repeated allocation without release.
  - Buffer overflow, in which a program tries to store data past the end of allocated storage. This may or may not lead to an access violation or storage violation. These bugs may form a security vulnerability.
  - Excessive recursion which — though logically valid — causes stack overflow.
  - Use-after-free error, where a pointer is used after the system has freed the memory it references.
  - Double free error.

# Common types of Software faults III

- Multi-threading programming faults
  - Deadlock, where task A can't continue until task B finishes, but at the same time, task B can't continue until task A finishes.
  - Race condition, where the computer does not perform tasks in the order the programmer intended.
  - Concurrency errors in critical sections, mutual exclusions and other features of concurrent processing. Time-of-check-to-time-of-use (TOCTOU) is a form of unprotected critical section.
- Performance faults
  - Too high computational complexity of algorithm.
  - Random disk or memory access.

# Common types of Software faults IV

- Interfacing faults
  - Incorrect API usage.
  - Incorrect protocol implementation.
  - Incorrect hardware handling.
  - Incorrect assumptions of a particular platform.
  - Incompatible systems.
- Team working faults
  - Unpropagated updates;
    - e.g. programmer changes "myAdd" but forgets to change "mySubtract", which uses the same algorithm.
    - Comments out of date or incorrect: many programmers assume the comments accurately describe the code.
  - Differences between documentation and the actual product.

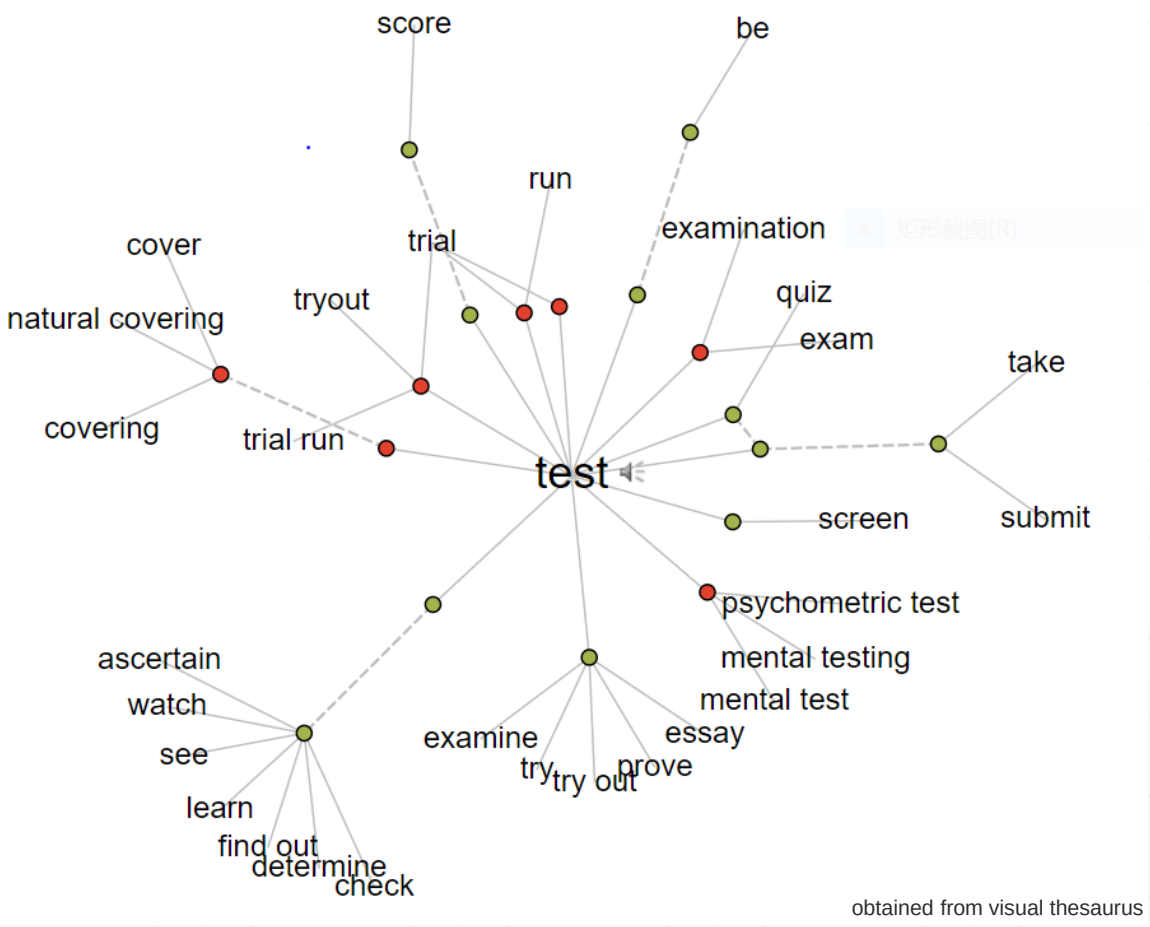


# Causes of Software Errors

- Human factor
- Communication failure
- Unrealistic development timeframe
- Poor design logic
- Poor coding practices
- Lack of version control
- Buggy third-party tools
- Lack of skilled testing
- Last minute changes

# Software Testing

# Software Testing




obtained from visual thesaurus

**Static:** Concerned with analysis of the static system representation to discover problems

- May be supplement by tool-based document and code analysis
- Code and document inspections
- Reviews, walkthroughs, or inspections

**Dynamic:** Concerned with exercising and observing product behaviour

- The system is executed with test data and its operational behaviour is observed

# Code Inspections

- Involves examining the source code with the aim of discovering anomalies and defects
- Intended for defect *detection,* not correction
- Very effective technique for discovering errors
- Saves time and money – the earlier in the development process an error is found, the better

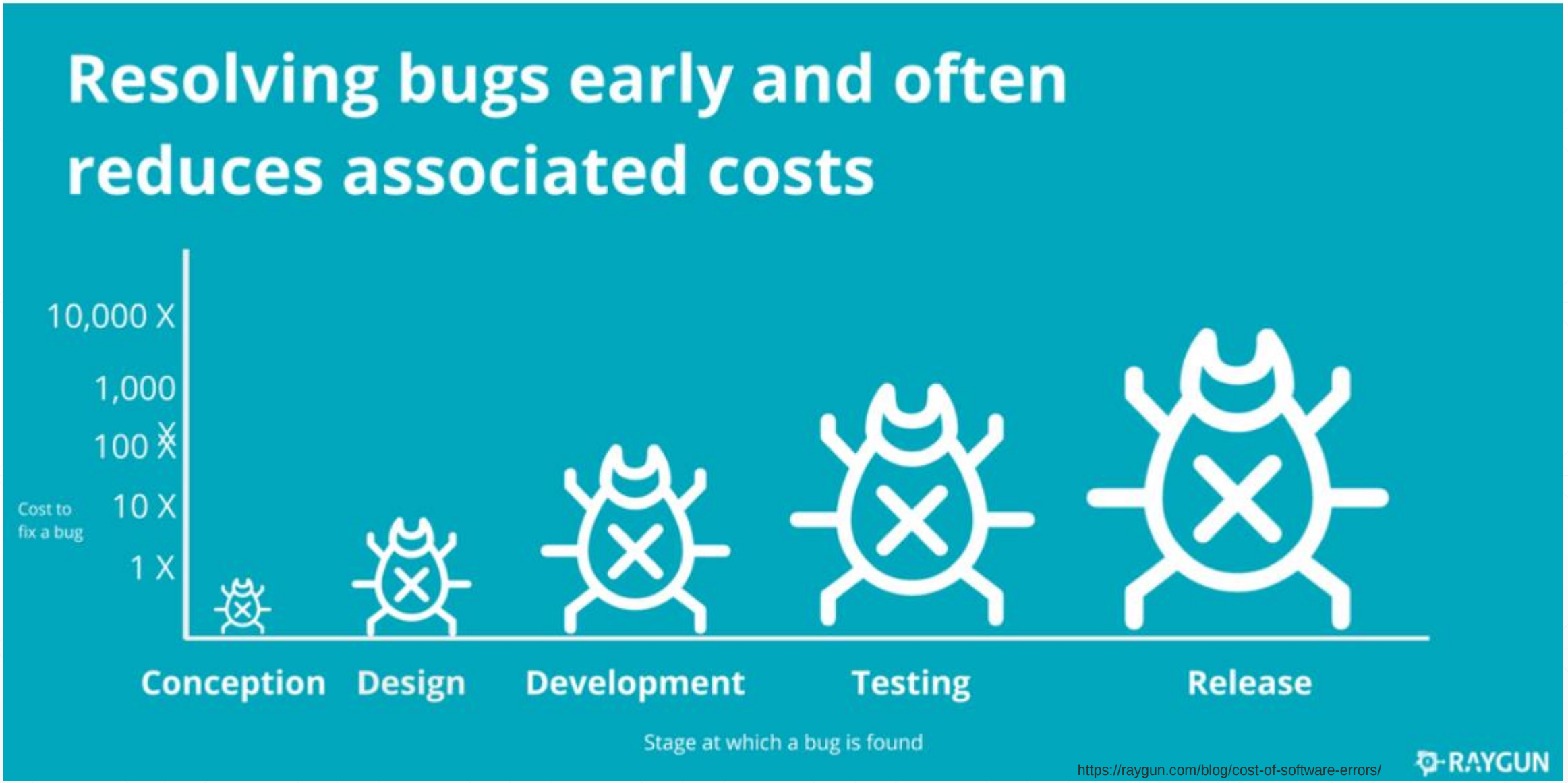
# Inspection Checklist

| Fault class | Static analysis check |
|---|---|
| Data faults | Variables used before initialization<br>Variables declared but never used<br>Variables assigned twice but never used between assignments<br>Possible array bound violations<br>Undeclared variables |
| Control faults | Unreachable code |
| Input/output faults | Variables output twice with no intervening assignment |
| Interface faults | Parameter type mismatches<br>Parameter number mismatches<br>Non-usage of the results of functions<br>Uncalled functions and procedures |
| Storage management faults | Unassigned pointers<br>Pointer arithmetic |
| Exception management faults | Have all possible error conditions been taken into account? |

# Software Testing in SWEBOK V3.0

| Table I.1. The 15 SWEBOK KAs |
|---|
| Software Requirements |
| Software Design |
| Software Construction |
| Software Testing |
| Software Maintenance |
| Software Configuration Management |
| Software Engineering Management |
| Software Engineering Process |
| Software Engineering Models and Methods |
| Software Quality |
| Software Engineering Professional Practice |
| Software Engineering Economics |
| Computing Foundations |
| Mathematical Foundations |
| Engineering Foundations |

- Software testing consists of the *dynamic* verification that a program provides *expected* behaviors on a finite set of test cases, suitably *selected* from the usually infinite execution domain.
- Testing is no longer seen as an activity that starts only after the coding phase is complete … Software testing is *pervasive throughout the entire* development and maintenance *life cycle .*


Resolving bugs early and often reduces associated costs
10,000 X
1,000
100 X
10 X
1 X
Cost to fix a bug
Conception
Design
Development
Testing
Release
Stage at which a bug is found
https://raygun.com/blog/cost-of-software-errors/
RAYGUN

# Why Do We Test Software ?

A tester's goal is to eliminate faults as early as possible

- Improve quality
- Reduce cost
- Preserve customer satisfaction

Testing can only show the presence of failures

Not their absence

# 15 min break

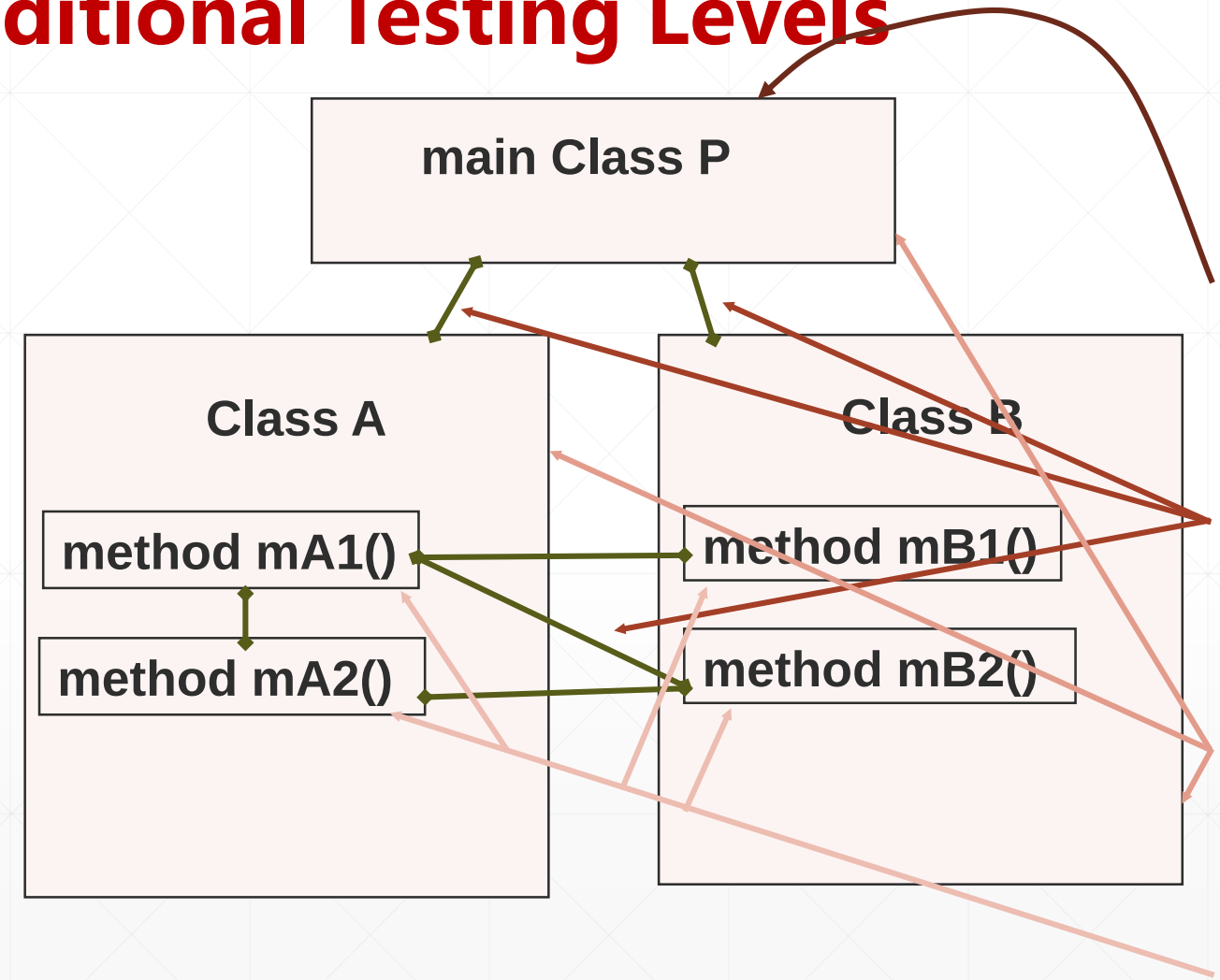
# Testing Levels

# Testing Levels

| Level | Summary |
|---|---|
| Unit Testing | A level of the software testing process where individual units of a software are tested. The purpose is to validate that each unit of the software performs as designed. |
| Integration Testing | A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. |
| System Testing | A level of the software testing process where a complete, integrated system is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements. |
| Acceptance Testing | A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery. |

Adopted from http://softwaretestingfundamentals.com/software-testing-levels/

# Traditional Testing Levels




Acceptance testing : Is the software acceptable to the user?

System testing : Test the overall functionality of the system

Integration testing : Test how modules interact with each other

Module testing (developer testing) : Test each class, file, module, component

Unit testing (developer testing) : Test each unit (method) individually

# Unit Testing

Unit Testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently tested to determine whether they are fit for use.

- Unit can be: a line of code, a  method, or a class Generally though, smaller is better
- What to test:
  - Algorithms and logic
  - Data structures (global and local)
  - Interfaces
  - Independent paths
  - Boundary conditions
  - Error handling
  - Others

# Unit Testing

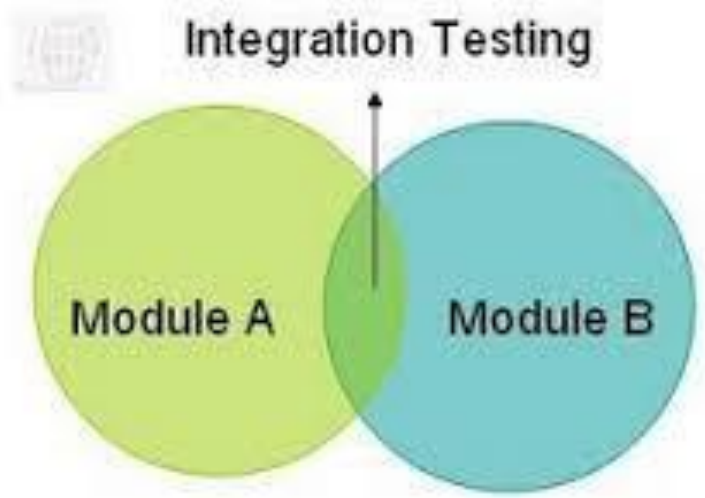
Goal: Isolate each part of the program and show that the individual parts are correct

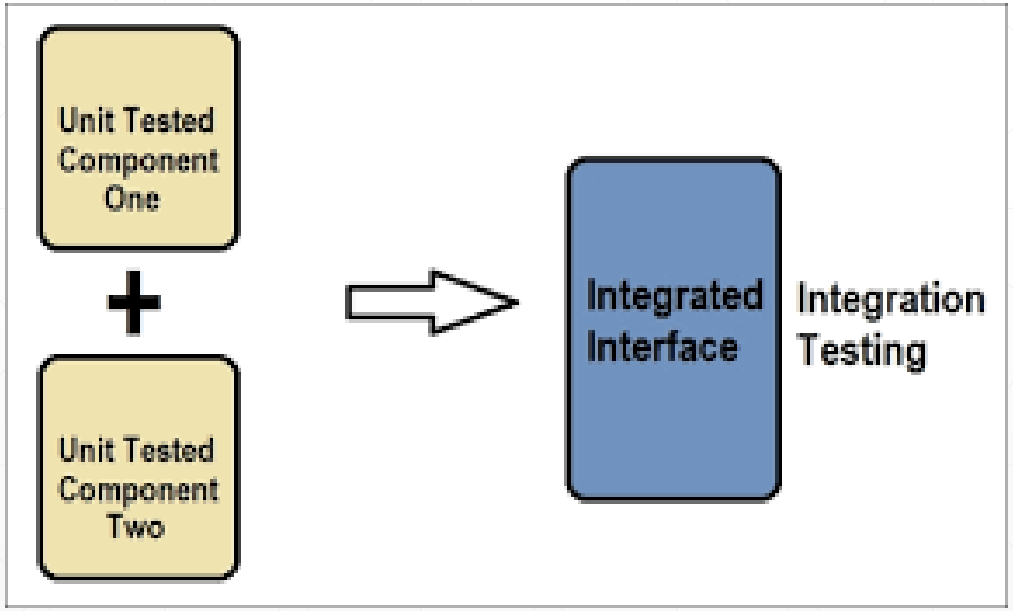
## Characteristics:

- Run completely by itself, without any human input. Unit testing is about automation.
- Determine by itself whether the function it is testing has passed or failed, without a human interpreting the results.
- Run in isolation, separate from any other test cases (even if they test the same functions). Each test case is an island
- Tests written in same language as the code
- Rapid feedback, can find problems early in the development cycle;

# Integration Testing

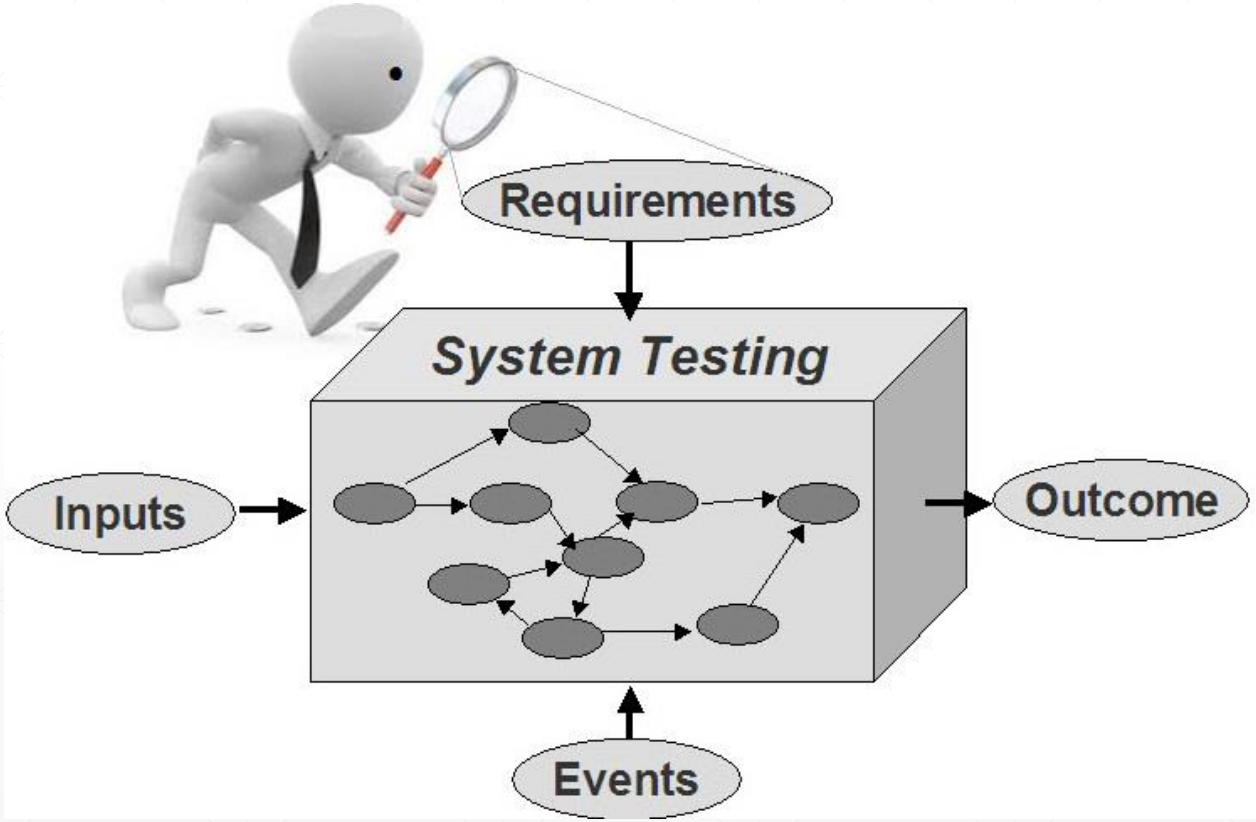
- Integration testing is the phase in software testing in which individual software modules are combined and tested as a group



# System Testing

- Determine if the software meets all of the requirements defined in the SRS



# Acceptance Testing

- Similar to system testing except that customers are present or directly involved.
- Usually the tests are developed by the customer
- Further categorized into Alpha and Beta Testing

# Testing Types

# Software Testing Types

- **Functional Testing**: the system is tested against the functional requirements/specifications.
- **Performance Testing**: intends to determine how a system performs in terms of responsiveness and stability under a certain load.
- **Security Testing**: intends to uncover vulnerabilities of the system and determine that its data and resources are protected from possible intruders.
- **Regression testing**: intends to ensure that changes (enhancements or defect fixes) to the software have not adversely affected it.
- **Compliance Testing/conformance testing/regulation testing/standards testing**: intends to determine the compliance of a system with internal or external standards.

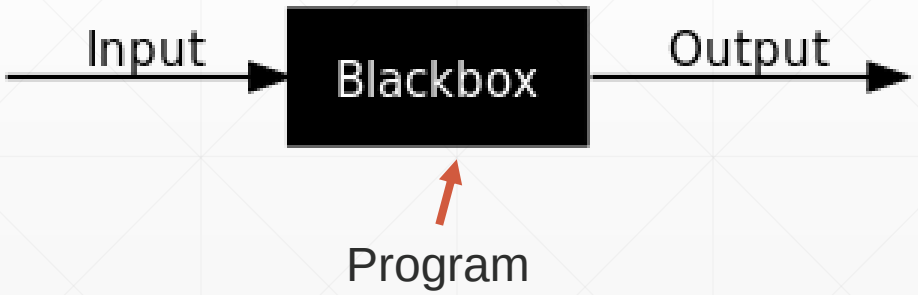
# Testing Methods

# Software Testing Methods

- Black Box Testing
- White Box Testing
- Gray Box Testing
- Agile Testing
- Ad Hoc Testing

# Black Box Testing I

- also known as Behavioral Testing,
- a software testing method in which the internal structure/design/ implementation of the item being tested is not known to the tester.
- can be functional or non-functional, though usually functional.



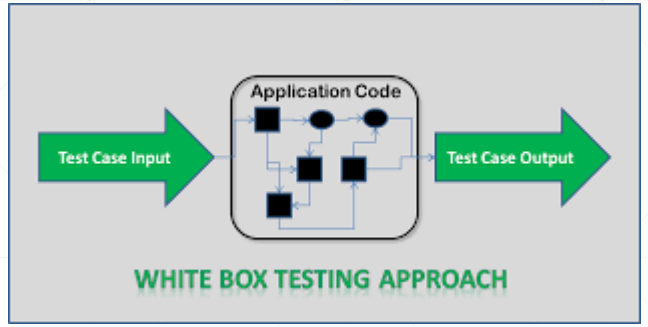
# Black Box Testing II

- Black Box Testing method is applicable to the following levels of software testing:
  - Integration Testing
  - System Testing
  - Acceptance Testing
- The higher the level, and hence the bigger and more complex the box, the more black-box testing method comes into use.

# Black Box Testing III

- Advantages
  - Tests are done from a user’ s point of view, help in exposing discrepancies in the specifications.
  - Tester need not know programming languages or how the software has been implemented.
  - Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
  - Test cases can be designed as soon as the specifications are complete.
- Disadvantages
  - Only a small number of possible inputs can be tested and many program paths will be left untested.
  - Without clear specifications, which is the situation in many projects, test cases will be difficult to design.
  - Ever wondered why a soothsayer closes the eyes when foretelling events? So is almost the case in Black Box Testing.

# White Box Testing I




- also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing
- a software testing method in which the internal structure/design/ implementation of the item being tested is known to the tester.
- The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential.
- White box testing is testing beyond the user interface and into the nitty-gritty of a system.

# White Box Testing II

- White Box Testing method is applicable to the following levels of software testing:
  - Unit Testing: For testing paths within a unit.
  - Integration Testing: For testing paths between units.
  - System Testing: For testing paths between subsystems.
- However, it is mainly applied to Unit Testing.

# White Box Testing III

- Advantages
  - Testing can be commenced at an earlier stage.
  - Testing is more thorough, with the possibility of covering most paths.
- Disadvantages
  - Since tests can be very complex, highly skilled resources are required, with a thorough knowledge of programming and implementation.
  - Test script maintenance can be a burden if the implementation changes too frequently.
  - Since this method of testing is closely tied to the application being tested, tools to cater to every kind of implementation/platform may not be readily available.

# Differences between BBT and WBT

| Criteria | Black Box Testing | White Box Testing |
|---|---|---|
| *Definition* | Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester | White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester. |
| *Levels Applicable To* | Mainly applicable to higher levels of testing: Acceptance Testing<br>System Testing | Mainly applicable to lower levels of testing: Unit Testing<br>Integration Testing |
| *Responsibility* | Generally, independent Software Testers | Generally, Software Developers |
| *Programming Knowledge* | Not Required | Required |
| *Implementation Knowledge* | Not Required | Required |
| *Basis for Test Cases* | Requirement Specifications | Detail Design |

# Grey Box Testing

- a software testing method which is a combination of Black Box Testing method and White Box Testing method.
- In Black Box Testing, the internal structure of the item being tested is unknown to the tester and in White Box Testing the internal structure is known.
- In Gray Box Testing, the internal structure is partially known. This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level.
- Though Gray Box Testing method may be used in other levels of testing, it is primarily used in Integration Testing.

# Agile Testing

- a method of software testing that follows the principles of agile software development.
- satisfy the customer through the early and continuous delivery of valuable software
- still need all those software testing methods, types, and artifacts (but at varying degrees of priority and necessity).
- however, completely throw away that traditional attitude and embrace the agile attitude.

# Ad hoc Testing

- also known as Random Testing or Monkey Testing,
- a method of software testing without any planning and documentation.
- defects found using this method are more difficult to reproduce (since there are no written test cases), sometimes very interesting defects are found which would never have been found if written test cases existed and were strictly followed.
- This method is normally used during Acceptance Testing.

# 5 min break

# Coverage Criteria

# Coverage Criteria

- Even small programs have too many inputs to fully test them all
  - **private static double** computeAverage (int A, int B, int C)
  - On a 32-bit machine, each variable has over 4 billion possible values
  - Input space might as well be infinite
- Testers search a huge input space
  - Trying to find the fewest inputs that will find the most problems
- Coverage criteria give structured, practical ways to search the input space
  - Search the input space thoroughly
  - Not much overlap in the tests

# Coverage Criteria

- Basic Coverage Criteria
  - **Function coverage** – Has each function (or subroutine) in the program been called?
  - **Statement coverage** – Has each statement in the program been executed?
  - **Branch coverage** – Has each branch of each control structure (such as in *if* and *case statements*) been executed? For example, given an *if* statement, have both the true and false branches been executed? Another way of saying this is, has every edge in the program been executed?
  - **Condition coverage** (or predicate coverage) – Has each Boolean sub-expression evaluated both to true and false?
  - **Decision coverage** – a combination of function coverage and branch coverage
  - …

# Advantages of Coverage Criteria

- Provide traceability from software artifacts to tests
  - Source, requirements, design models, …
- Make regression testing easier
- Gives testers a "stopping rule" … when testing is finished
- Can be well supported with powerful tools

# Testing Activities

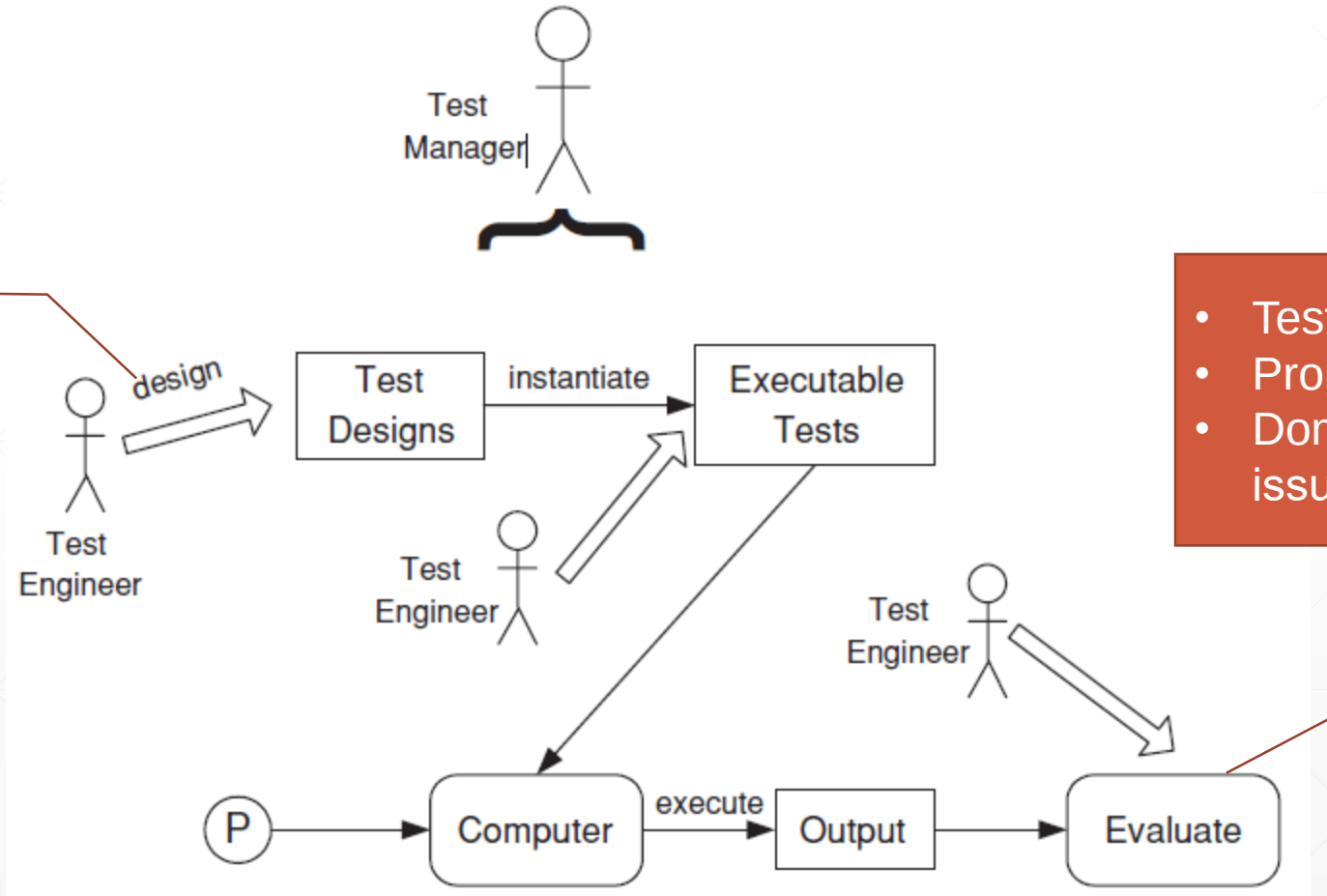
# Software Testing Activities

- Test Engineer : An IT professional who is in charge of one or more technical test activities
  - Designing test
  - automate test
  - Running test scripts
  - Analyzing results and reporting results to developers and managers

- Test Manager : In charge of one or more test engineers
  - Sets test policies and processes
  - Interacts with other managers on the project
  - Otherwise supports the engineers

# Testing Activities

- Type of testing
- Testing techniques
- Coverage criteria
- Tools to use
- …




- Test pass or fail?
- Proper criteria?
- Domain specific issues…

# Types of Test Activities

- Testing can be broken up into four general types of activities
  1. **Test Design**
  2. **Test Automation**
  3. **Test Execution**
  4. **Test Evaluation**
- Each type of activity requires different skills, background knowledge, education and training
- No reasonable software development organization uses the same people for requirements, design, implementation, integration and configuration control

# 1. Test Design—(a) Criteria-Based

- This is the most technical job in software testing
- Requires knowledge of :
  - Discrete math
  - Programming
  - Testing
- This is intellectually stimulating, rewarding, and challenging
- Test design is analogous to software architecture on the development side
- Using people who are not qualified to design tests is a sure way to get ineffective tests

**Design test values to satisfy coverage criteria or other engineering goal**

# 1. Test Design—(b) Human-Based

- This is much harder than it may seem to developers
- Criteria-based approaches can be blind to special situations
- Requires knowledge of :
  - Domain, testing, and user interfaces
- Requires almost no traditional CS
  - A background in the domain of the software is essential
  - An empirical background is very helpful (biology, psychology, …)
  - A logic background is very helpful (law, philosophy, math, …)
- This is intellectually stimulating, rewarding, and challenging
  - But not to typical CS majors – they want to solve problems and build things

**Design test values based on domain knowledge of the program and human knowledge of testing**

# 2. Test Automation

- This is slightly less technical
- Requires knowledge of programming, but very little theory
- Often requires solutions to difficult problems related to observability and controllability
- Can be boring for test designers
- Programming is out of reach for many domain experts
- Who is responsible for determining and embedding the expected outputs ?
  - Test designers may not always know the expected outputs
  - Test evaluators need to get involved early to help with this

**Embed test values into executable scripts**

# 3. Test Execution

- This is easy – and trivial if the tests are well automated
- Requires basic computer skills
  - Interns
  - Employees with no technical background
- Asking qualified test designers to execute tests is a sure way to convince them to look for a development job
- If, for example, GUI tests are not well automated, this requires a lot of manual labor
- Test executors have to be very careful and meticulous with bookkeeping

**Run tests on the software and record the results**

# 4. Test Evaluation

- This is much harder than it may seem
- Requires knowledge of :
  - Domain
  - Testing
  - User interfaces and psychology
- Usually requires almost no traditional CS
  - A background in the domain of the software is essential
  - An empirical background is very helpful (biology, psychology, …)
  - A logic background is very helpful (law, philosophy, math, …)
- This is intellectually stimulating, rewarding, and challenging
  - But not to typical CS majors – they want to solve problems and build things

**Evaluate results of testing, report to developers**

# Other Activities

- Test management : Sets policy, organizes team, interfaces with development, chooses criteria, decides how much automation is needed, …
- Test maintenance : Save tests for reuse as software evolves
  - Requires cooperation of test designers and automators
  - Deciding when to trim the test suite is partly policy and partly technical – and in general, very hard !
  - Tests should be put in configuration control
- Test documentation : All parties participate
  - Each test must document "why" – criterion and test requirement satisfied or a rationale for human-designed tests
  - Ensure traceability throughout the process
  - Keep documentation in the automa tests

# Organizing the Team

- A mature test organization needs only one test designer to work with several test automators, executors and evaluators
- Improved automation will reduce the number of test executors
  - Theoretically to zero … but not in practice
- Putting the wrong people on the wrong tasks leads to inefficiency, low job satisfaction and low job performance
  - A qualified test designer will be bored with other tasks and look for a job in development
  - A qualified test evaluator will not understand the benefits of test criteria
- Test evaluators have the domain knowledge, so they must be free to add tests that "blind" engineering processes will not think of
- The four test activities are quite different

**Many test teams use the same people for ALL FOUR activities !!**

# Software Failures Errors & Fault

- 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 Fault : A static defect in the software, root cause

# Failure Error & Fault -- Example

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

# Good Testing Practices I

- A good test case is one that has a high probability of detecting an undiscovered defect, not one that shows that the program works correctly
- It is impossible to test your own program
- A necessary part of every test case is a description of the expected result
- Write test cases for valid as well as invalid input conditions.
- Thoroughly inspect the results of each test

# Good Testing Practices II

- As the number of detected defects in a piece of software increases, the probability of the existence of more undetected defects also increases
- Assign your best people to testing
- Ensure that testability is a key objective in your software design
- Never alter the program to make testing easier
- Testing, like almost every other activity, must start with objectives

# To Conclude

- Software Quality
- Software Testing
  - Testing Levels
  - Testing Types
  - Testing Methods
  - Coverage Criteria
  - Testing Activities