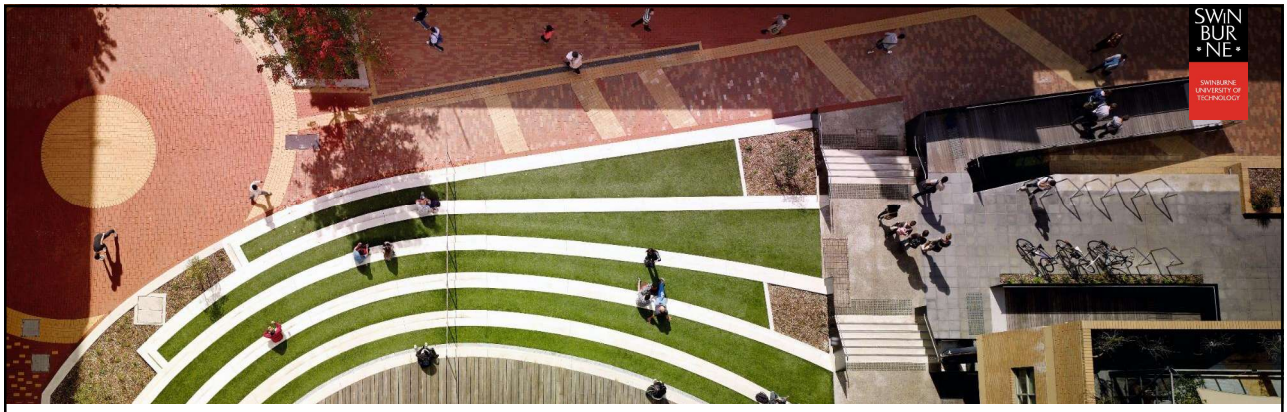


COS80022 – Software Quality and Testing

Week 4 – Software Testing Foundation

Outline

- Relevant terminology
- Basic concepts of testing
- Objectives
- Principles

Theoretical Foundation

Utopian Software Development

- In a perfect world, software developers deliver solutions that meet stated requirements in an efficient and timely manner.
- User Satisfaction = Compliant Product + Good Quality + Delivery within budget and schedule

– Pressman & Maxim

However …

Real World Software Development

- Most software solutions have bugs (or defects)
- A bug is an *unwanted* and *unintended* property of any piece of software, especially one that causes it to *malfunction* or *fail*

Quality Control in Software Development

- Quality control mechanisms are required to ensure that the functional and quality requirements are met
- Validation and Verification (V&V) provide this control mechanism
  - Ensure high quality software
  - Combine preventative and corrective measures

Validation and Verification

- Validation
  - "**Are we building the right product?**" (Boehm)
  - Does the system meet the customer's expectation
  - More on user side
- Verification
  - "**Are we building the product right?**" (Boehm)
  - Does the software conform to its specification?
  - More on product/process side

Boehm, B.W., 1984. Verifying and validating software requirements and design specifications. *IEEE software*, *1*(1), p.75.

Validation and Verification

- Validation
  - More general
  - Ensure that the software meets the customer's expectations
- Verification
  - Check that the software meets its stated functional and non-functional requirements

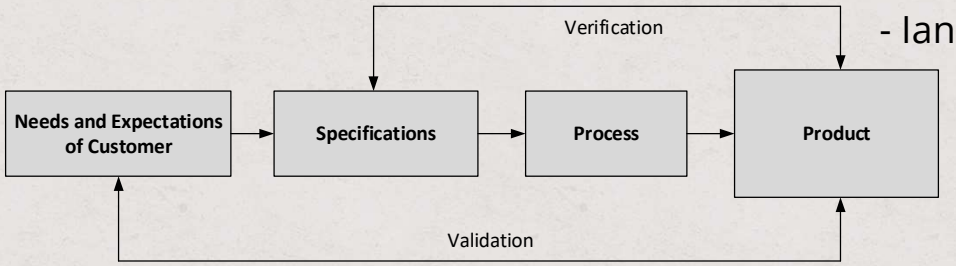
Validation and Verification

- Aim of V & V is to establish confidence that the system is 'fit for purpose'.
- Depends on system's purpose, user expectations and marketing environment
  - Software purpose
    - The level of confidence depends on how critical the software is to an organisation.
  - User expectations
    - Users may have low expectations of certain kinds of software.
  - Marketing environment
    - Getting a product to market early may be more important than finding defects in the program.

Validation and Verification

- *"Verification and validation (V&V) is the name given to the checking and analysis processes that ensure that software conforms to its specification and meets the needs of customers who are paying for that software."*

- Ian Sommerville



Sommerville, I., 2011. Software engineering 9th Edition. *ISBN-10, 137035152*.

Verification

- Is the software being built right?
- Does the implementation fully and correctly realise the specification?
- Do all components exhibit good workmanship, sufficient performance, and conform to applicable standards
- Verification addresses the extent of fulfilment of work
- Verification can be
  - Static
  - Dynamic

Approaches To Verification

- Static Checking:
  - Static analysis of source code using automated tool
  - Identify specific problems in software e.g. memory leak
  - Cannot find vulnerabilities introduced in the runtime environment
  - Can result in false positives and false negatives
- Dynamic Testing
  - Feed input to software and run it to see whether its behaviour is as expected
  - Dynamic analysis of running code
  - Impossible to cover all cases

Approaches To Verification

- Formal Proof
  - Use rigorous mathematical methods to prove that the program implements the specification
  - Limitations
    - Difficult to have a formal specification
    - The proof costs a lot of human efforts
- Software Inspection/Review
  - Static approach
  - Manually review the code to detect faults
  - Limitations
    - Hard to evaluate
    - Sometimes hard to get progress

Preferred Approach → Testing

- *"50% of my employees are testers, and the rest spend 50% of their time testing"*

  - Bill Gates, 1995

- More reliable than inspection, relatively cheap
  - In the old days, when testing was expensive, inspection was the major answer
- You get what you pay for (linear awards)
  - Compared to other three approaches
    - Inspection, static checking, formal proof

Validation

- Are we building the right product?
- Is the specification correct?
- Are all user requirements met?
- Validation addresses the extent of fulfilment of customer requirements, in any phase
- Techniques used
  - System testing and acceptance testing
  - Formal reviews

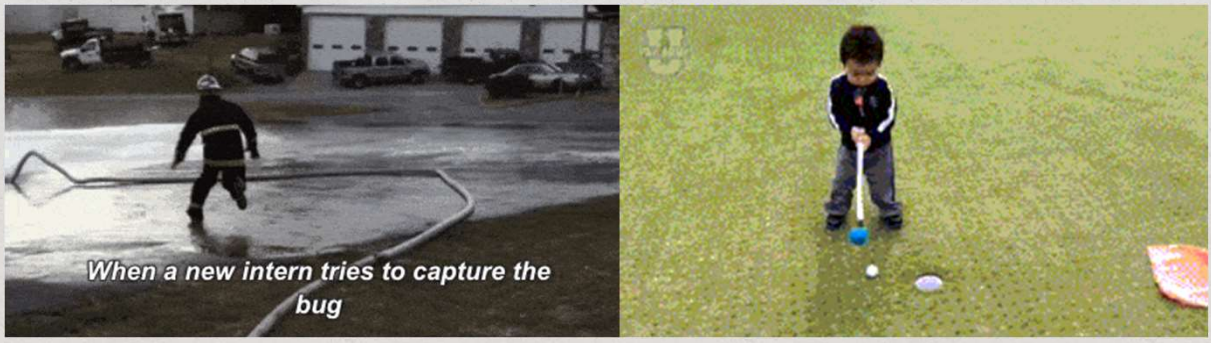
How To V & V Your Solution?

- Stare at your code for a while and convince yourself that it works
  - Easy to make mistakes
  - Easy to be lazy
  - Suffers from *tunnel vision*
  - Most software is tested this way
- Prove that it is correct
  - Difficult (very)
  - Not always possible
- Ship it to customers and wait for their complaints
  - Not very nice

How To V & V Your Solution?

- Testing and Debugging
- *Testing* – a means of detecting/revealing errors
- *Debugging* – a means of diagnosing and correcting the root causes of errors that have already been detected

Testing and Debugging



Terminology

- According to IEEE Software Quality Management standards
  - *Mistake:* human mistake results in manifestation of a fault during the execution of the system
  - *Fault or Defect:* an incorrect statement, step, process, or data definition in a computer program. This is generally caused by a design or coding mistake. A single fault can cause multiple failures.
  - *Failure:* a manifest observable violation against specification or client expectations by the system. A failure can be thought of as a symptom.
  - *Error:* observed difference between a computed result and the correct result i.e. Expected Outcome ≠ Observed Outcome
  - The term *bug* is used colloquially to mean one or all of the above

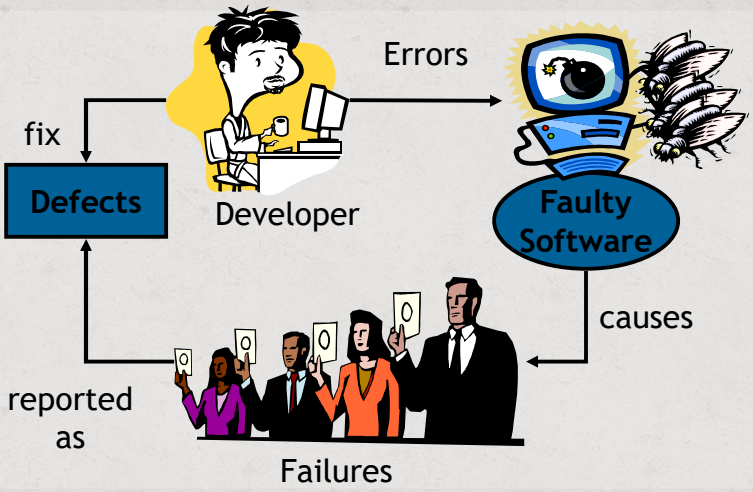
Mistake, Fault & Failure

- For example, developer makes a mistake
  - E.g. “>” instead of “>=“
- This results in code with some omissions or faults
- These faults manifest as a failure when the program is in production or test

Terminology

- ISTQB (International Software Testing Qualifications Board)
  - Mistake → Error
  - Fault ≡ Defect
  - Failure

Life Cycle of a Bug



Error, Defect and Failure

- When developing software, people make errors, these become defects (faults) in the software which then manifest themselves as failures when the software is run.
- One error may lead to several different defects, each of which in turn leads to several different failures.

Error, Defect and Failure

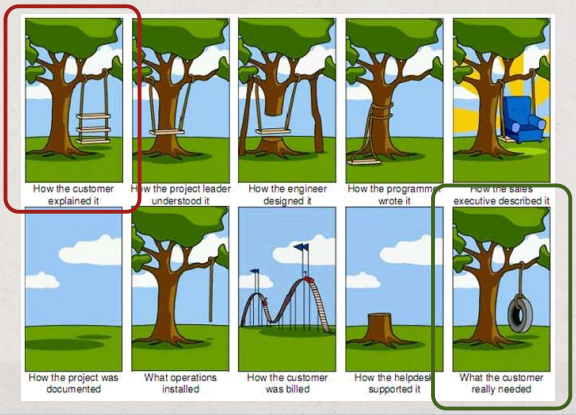
- Code to decide whether a triangle is right-angled.
  - if (a * a + b * b == c * c)
    cout << “right-angled” << endl;
  - if (a * a * a + b * b * b == c * c * c)
    cout << “right-angled” << endl;

Error, Defect and Failure

- Error: the programmer uses a wrong formula
- Defects: a * a → a * a * a; b * b → b * b * b; c * c → c * c * c.
- Failures: some triangles are mistakenly considered as non-right-angled; while some triangles are mistakenly considered a right-angled.

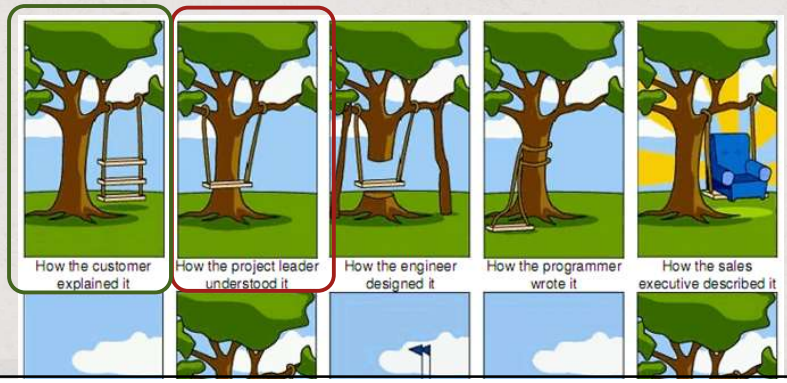
Why do we have bugs?

- *Users specify wrong requirements*
  - Correct business rule: *"Salary should be >= 50,000"*
  - User specification: *"Salary should be > $50,000"*



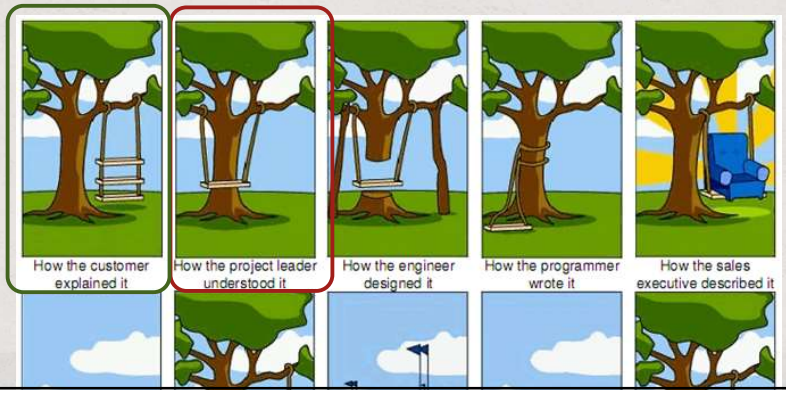
Why do we have bugs?

- *Developer misinterprets requirements*
  - Requirement: *"There should be no more than 100 students in the class"*
  - Interpretation: *"There should be more than 100 students in each class"*



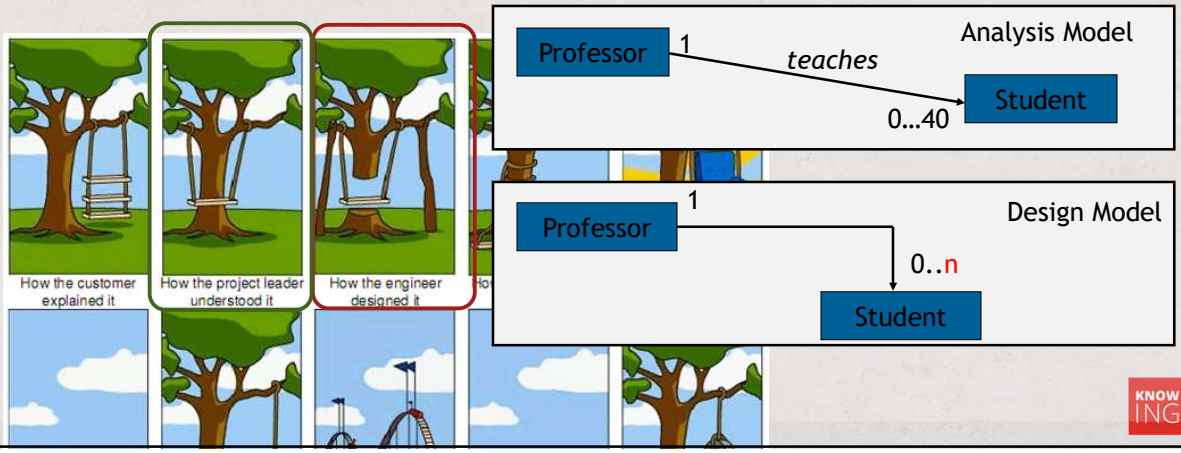
Why do we have bugs?

- *Requirements are incorrectly recorded*
  - Business Rule: "*First $6000* of income is not taxed. Next $6000 is taxed at 10%"
  - Recorded as: "*First $600* of income is not taxed. Next $6000 is taxed at 10%"



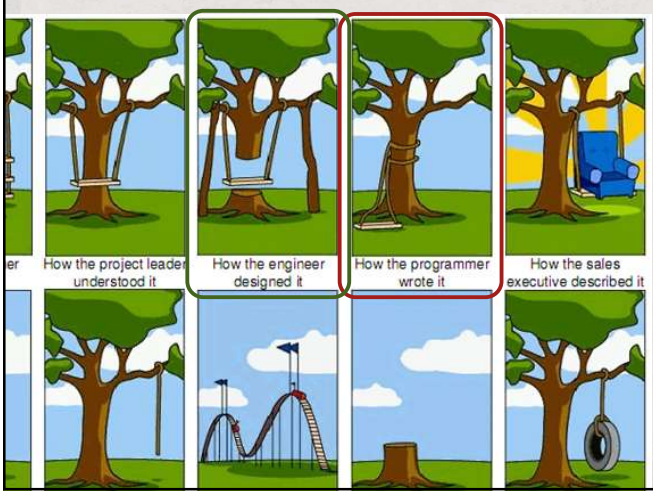
Why do we have bugs?

- *Design specifications are incorrect or inconsistent with requirements*

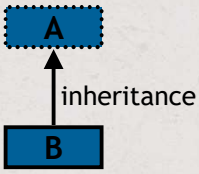


Why do we have bugs?

- *Program specification are incorrect or inconsistent with design*

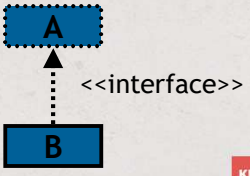


Design:



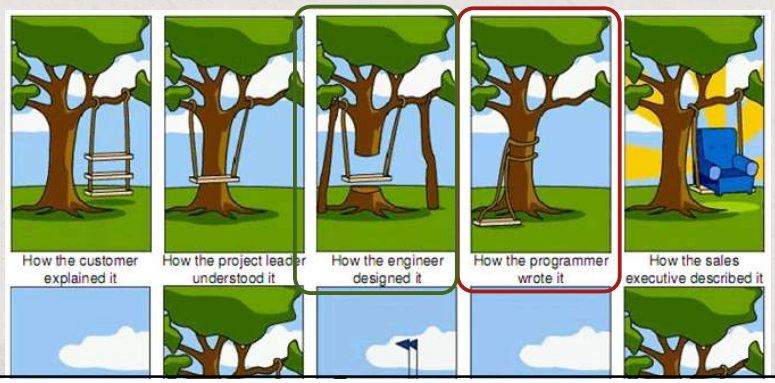
A is abstract class

Implementation:
Class A is declared as Interface
Class B implements realizing interface Class A



Why do we have bugs?

- *Programming mistakes in the coding phase*
- *Required: a>= 52*
- *Actual: a>=5*



Why do we have bugs?

- Mistake made during data entry or data translation
- Mistakes during data entry – database has invalid data

Requirement	
Salary	**Tax Rate**
6000	**0%**
6000-12000	**10%**
12000 >	**20%**

Entry Made As		
Min	**Max**	**Rate**
0	6000	0%
6000	12000	10%
12000	>	20%

Why do we have bugs?

- Testers make mistakes during testing
- The following represents a faulty test case

Salary	Expected Tax	Actual Tax	Output
$5000	$0	$50	PASS

Tester overlooks failure - reports a PASS instead of FAIL

Why do we have bugs?

- Bug fixes cause new bugs

Software Systems Context

- Most people have had an experience with software that did not work as expected
- Software that does not work correctly can lead to problems:
  - Loss of money
  - Loss of time
  - Damage to business reputation
  - Injury/death

Software Bugs → Financial Costs

- Japan's Hitomi Telescope
  - https://www.youtube.com/watch?v=Q1KrxdT5NcQ
- Confused about how it was oriented
  - Fire in wrong direction – accelerating rather than slowing
  - Spun faster and faster

Software Bugs → Human Costs

- Tesla FSD Beta 9.0
  - https://www.youtube.com/watch?v=uIsMA9qNwvM
- 11 scenarios
  - Doesn't recognise objects
  - Wrong lane

Defective Code - Thoughts

- Imagine a large system that has 1000 critical bugs
  - How many tests do we have to run to get these all out?
- According to the 2017 "Software Fail Watch" by Tricentis
  - Software failures caused the loss of over 1.7 trillion US dollars in 2017 alone
  - Affected the lives of about 3.7 billion people (half of the world population)

Defective Code - Statistics

- Studies* have shown the following about the Defect Rate:

Novice Developer (1-3 years experience)	Experienced Developer (3-7 years experience)	Expert Developer (over 7 years experience)
5-7 mistakes per 100 LOC	1 mistake per 100 LOC	0.5 mistake per 100 LOC

LOC = Lines of Code

* - Stats are applicable to Java

Why Do We Need Testing?

- Real programmers do not test (or do they?)
  - I want to get this done fast – testing will slow me down
  - I started programming when I was 2. Don't insult me by testing my perfect code!
  - Testing is for incompetent programmers
  - "Most of the functions in Graph.java, as implemented, are one or two line functions that solely rely on functions in HashMap or HashSet. I am assuming that *these functions work perfectly*, and thus there is really no need to test them"

Testing

- Testing and code reviews/inspection are the most common quality-assurance methods
- Testing is a practice supported by a wealth of industrial and academic research and by commercial experience
- "*Testing is the process of executing a program with the intention of finding bugs*" – Myers (The Art of Software Testing)
- "*Testing can show the presence of bugs but never their absence*" – Dijkstra (Notes on Structured Programming - http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF)

Software Testing

Testing – History

What is Testing?

- IEEE definition:

  Software testing is the process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item.

What is Testing?

- Testing: by experiment,
  - find faults in software
  - establish quality of software
- A “good” test:
  - finds at least one fault
  - gives quality judgement with maximum confidence with minimum effort

What is Testing?

- Dijkstra:
  - Testing can show the presence of bugs, but not the absence
- Beizer:
  - 1st law: Every method you use to prevent or find bugs leaves a residue of subtler bugs, for which other methods are needed
  - 2nd law: Software complexity grows to the limits of our ability to manage it

What is Testing?

Testing – History

- ~~PHASE 0: There's no difference between testing and debugging. Other than in support of debugging, testing has no purpose.~~
- ~~PHASE 1: The purpose of testing is to show that the software works.~~
- PHASE 2: The purpose of testing is to show that the software **doesn't work**.
- PHASE 3: The purpose of testing is not to prove anything, but to **reduce the perceived risk** of not working to an acceptable value.
- PHASE 4: Testing is not an act. It is a mental discipline that results in **low-risk software** without much testing effort.

Why Testing?

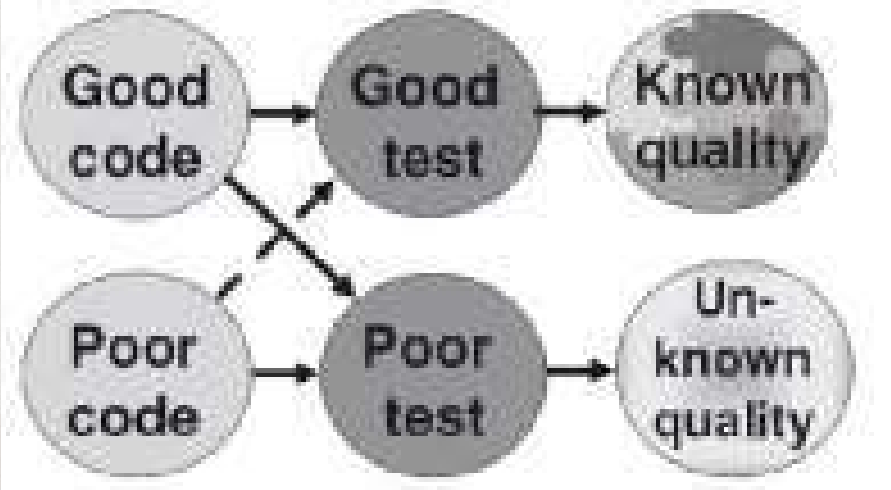
- What testing does and therefore the immediate purpose of testing is getting information about the product under testing.
- Testing is the intelligence office of the company.

Why Testing?

- Examples of data gathered in testing
  - Number of passed test cases
  - Coverage of the performed test
  - Number and types of failures
  - Defects corrected over time
  - Root causes of the failures

Why Testing?

- Value of decision improvement



Why Testing?

- Value of process improvement
  - The information gained from testing is invaluable in the analysis of how well processes fit and serve the organization.
  - The results of such analysis can be used to identify the process that could be the subject for process improvement.

Fundamental Problems

- Reliable test set problem
  - exhaustive testing?
  - reliable for any given program?
- Oracle problem
  - test oracle
  - difficult to verify test results

Reliable Test Set Problem

55

Reliable Test Set Problem

- Infinite monkey theorem
  - A monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will almost surely type a given text, such as the complete works of William Shakespeare.
- Unfortunately, we, as testers, are never given an infinite amount of time!

56

Reliable Test Set Problem

- *p* : a program
- *D* : the input domain (all possible inputs to *p*)
- If we test *p* on every element of *D*, and the results are correct, then *p* is a correct program
- not practical when *D* has a large or infinite size.

Reliable Test Set Problem

- We want to find a subset of test cases *T* from *D*
  - If *p*(*T*) is correct, then *p*(*D*) is correct
  - We say *T* is a **Reliable Test Set**.

Reliable Test Set Problem

- Problem: In general, *T* (of finite size) for any given program cannot be constructed effectively, unless *T* = *D* (known as "exhaustive testing")
- In other words: reliable test sets are not attainable in general
- As a result, testing cannot prove program correctness in most situations.

Oracle Problem

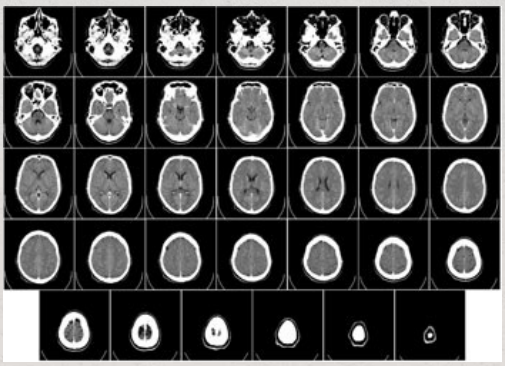
- Test oracle
  - Expected output
  - A way to check the correctness of program output
- $ax^2 + bx + c = 0$
  - $x = [-b \pm (b^2 - 4ac)^{1/2}] / 2a$
  - Replace x by the calculated results

Oracle Problem

- Oracle does not always exist
  - It is too expensive to apply oracle in practice
  - Oracle problem affects the testing effectiveness

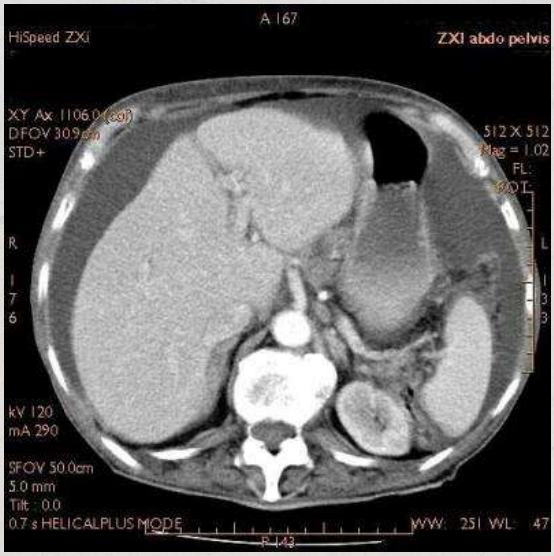
61

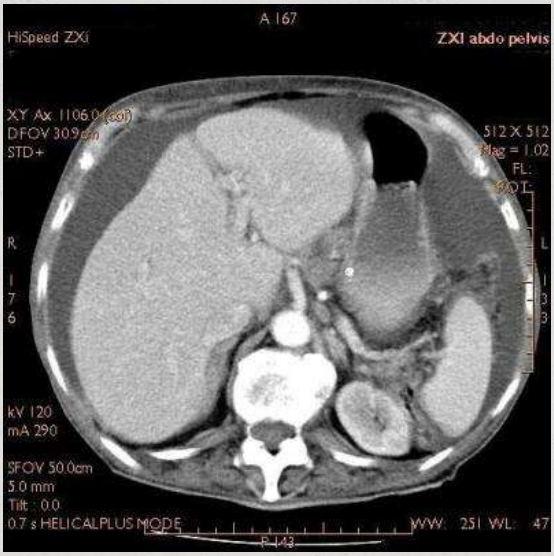
Oracle Problem



62

Oracle Problem



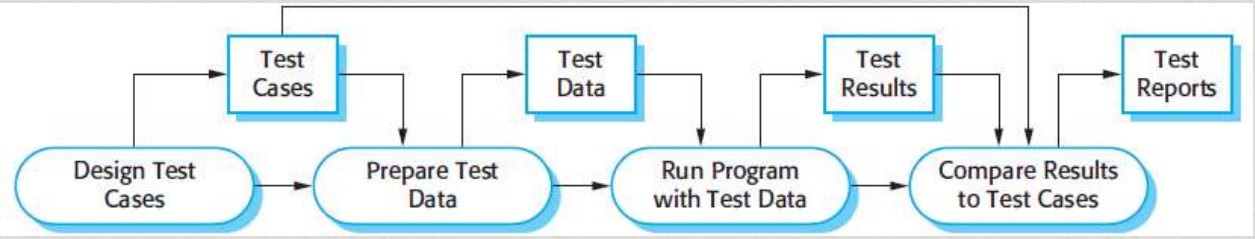


63

Oracle Problem

64

How Do We Test?



Testing Activities

- Test Planning
- Analysing
- Designing
- Implementing tests
- Reporting test progress and results
- Evaluating the quality of a test object
  - To be further elaborated in the next lecture

Testing Activities

- Static testing
  - Testing of a component or system at specification or implementation level without execution of that software
    - To be covered in Week 10's lecture
- Dynamic testing
  - Process of evaluating behaviour of a system or component during execution
    - To be taught in Weeks 7, 9 & 11

How Do We Test?

- Test case
  - A set of input data and expected results
  - Purpose: causing failure and detecting defects
- Five attributes:
  - Name
  - Location
  - Input
  - Oracle
  - Log

How Do We Test?

Attributes	Description
`name`	Name of test case
`location`	Full path name of executable
`input`	Input data or commands
`oracle`	Expected test results against which the output of the test is compared
`log`	Output produced by the test

Where Do We Test?

- Basically

Everywhere

– Next Lecture: Testing in Software Development Lifecycle

Typical Test Objectives

- Evaluate work products
- Verify specified requirements
- Validate test objects
- Build confidence
- Prevent defects
- Find failures and defects
- Provide information
- Reduce level of risk
- Comply with contractual/legal requirements

Evaluate Work Products

- Work products
  - Requirements
  - User stories
  - Design
  - Code

Verify specified requirements

- Whether all specified requirements have been fulfilled by building traceability to test cases
- **Traceability Matrix**
  - a document usually in the form of a table
  - to assist in determining the completeness of a relationship by correlating any two baselined documents using a many-to-many relationship comparison

Traceability Matrix

Sno	Req ID	Req Desc	TC ID	TC Desc	Test Design	Test Designer	UAT Test Req?	Test Execution			Defects?	Defect ID	Defect Status	Req Coverage Status
								Test Env	UAT Env	Prod Env				
1	Req01	Login to the Application	TC01	Login with Invalid Username and valid password	Completed	XYZ	No	Passed	No Run	No Run	None	None	N/A	Partial
2			TC02	Login with Valid Username and invalid password	Completed	YZA	No	Passed	No Run	No Run	None	None	N/A	Partial
3			TC03	Login with valid credentials	Completed	XYZ	Yes	Passed	Passed	No Run	Yes	DFCT001	Test OK	Partial

Sheet1 Sheet2 Sheet3

Traceability Matrix

"After the definitions of use cases, a spreadsheet shall be created by project manager and maintained by developer. In the spreadsheet, every use case shall be cross-referenced to its corresponding code and test cases, to ensure the completeness of development and V&V."

– An anonymous project manager

Validate Test Objects

- Whether the test object is complete and works as the users and other stakeholders expect

Build confidence

- In the quality level of the test object

Prevent Defects

- Review of documents (e.g. requirements) and the identification and resolution of issues can help to prevent defect appearing in the code

Find Failures and Defects

- The **main objective**
- To cause as many failures as possible so that defects in the code are identified and fixed

79

Provide Information

- Sufficient information to stakeholders to allow them to make informed decisions

80

Reduce Level of Risk

- Risk of inadequate software quality
  - Previously undetected failures occurring in operation

Comply with Contractual/Legal Requirements

- To verify the test object's compliance with such requirements or standards

Test Objectives Context

- Vary depending upon the context of
  - Component or system being test
  - Test level
  - Software development lifecycle model
- For example,
  - Component level: to find as many failures as possible
  - Acceptance level: working at expected and satisfying requirements

Testing Contributes to Success

Seven Principles

Seven Testing Principles

1. Testing show the presence of defect, not their absence
2. Exhaustive testing is impossible
3. Early testing saves time and money
4. Defect cluster together
5. Beware of the pesticide paradox
6. Testing is context dependent
7. Absence of errors is a fallacy

Principle 1

- Show that defects are present
- Cannot prove that there are no defects
- Reduces the probability of undiscovered defects remaining in the software
- Even if no defects are found, it is no proof of correctness

87

Principle 2

- Testing everything is not feasible
- Instead of exhaustive testing, risk analysis and priorities should be used to focus testing efforts.

88

Risk Analysis

- Risk-based approach is a way to prioritise development tasks
- Two dimensions:
  - from the technical perspective: the probability of a function failure
  - from the business perspective: the impact of a function failure

Risk Analysis

RISK ASSESSMENT MATRIX				
SEVERITY / PROBABILITY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
Frequent (A)	High	High	Serious	Medium
Probable (B)	High	High	Serious	Medium
Occasional (C)	High	Serious	Medium	Low
Remote (D)	Serious	Medium	Medium	Low
Improbable (E)	Medium	Medium	Medium	Low
Eliminated (F)	Eliminated			

Principle 3

- To detect defects early, testing activities shall be started as early as possible in the SDLC
- Early testing is sometimes referred to as **shift left**
- Testing early in the SDLC helps reduce or eliminate costly changes

Principle 4

- A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for the most of the operational failures
- Predicted defect clusters, and the actual observed defect clusters in test or operation, are an important input into a **risk analysis** used to focus the test effort.

Principle 5

- If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new defects
- To overcome this, test cases need to be regularly reviewed and revised. New and different tests also need to be written to exercise different parts of the software/system to find potentially more defects.

Principle 6

- Testing is done differently in different contexts
  - Safety critical software is tested differently from an e-commerce mobile app
  - Testing in an agile project to done differently than testing in a sequential lifecycle project

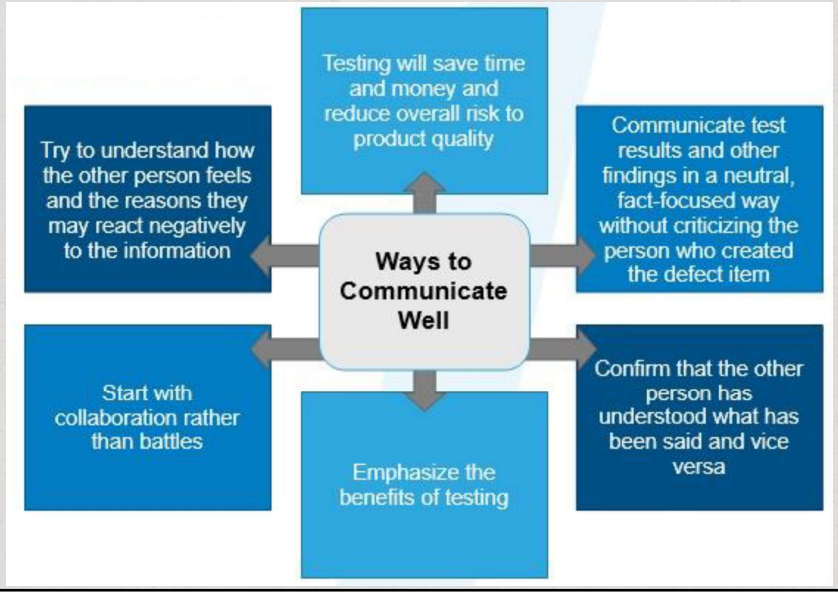
Principle 7

- Some organisations expect that testers can run all possible tests and find all possible defects, but principles 2 and 1, respectively, tell us that this is impossible
- It is a fallacy (a mistaken belief) to expect that just finding and fixing a large number of defects will ensure the success of a system

Human Psychology in Testing

- Identifying defects during a static test such as requirements review or user story refinement session
- Identifying failures during dynamic test execution, may be perceived as criticism of the product and of its author
- An element of human psychology called confirmation bias can make it difficult to accept information that disagrees with currently held beliefs

Communication Skills



97

Mindsets

98

Tester Mindset

- Curiosity
- Professional pessimism
- A critical eye
- Attention to detail
- Motivation for good and positive communications and relationships
- Tends to grow and mature as the test gains experience

Independent Testers

- With the right mindset, developers are able to test their own code
- Having some of the test activities done by independent testers increases defect detection effectiveness, which is particularly important for large, complex, or safety-critical systems
- Independent testers bring a perspective which is different than that of the work product authors since they have different cognitive biases form the authors.

Summary

- Terminology:
  - Verification vs. Validation
  - Error, Defect and Failure
- Testing:
  - What
  - Why

References

- Sommerville, I. *Software Engineering* (Chapter 8)
- Bruegge, B. & Dutoit A.H. *Object-Oriented Software Engineering Using UML, Patterns, and Java* (Chapter 11)
- International Software Testing Qualifications Board. *ISTQB Foundation Level (Core) Syllabus*

Next

- Weekly task
  - Design test cases
- Next week
  - Testing Process
  - Testing in SDLC