# LECTURE 6 – TESTING (PART 1)

Master of Applied Computing

COMP-8117 : Advanced Software Engineering Topics

Dr. Aznam Yacoub – aznam.yacoub@uwindsor.ca
School of Computer Science

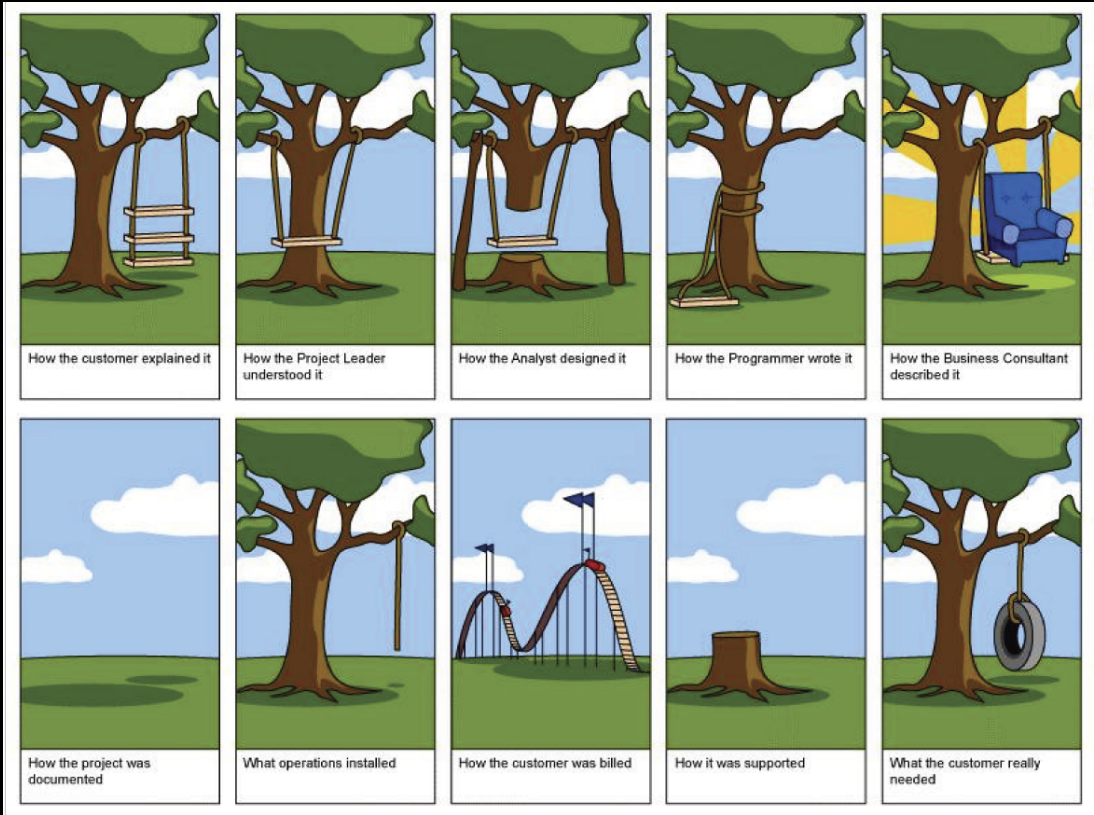
# SCHEDULE

- Introduction
- Tests : Philosophy and Methods
- Introduction to Test-Driven Development
- Conclusion

Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# REMIND OUR ADVENTURE




 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# REMIND OUR ADVENTURE - BEFORE

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# REMIND OUR ADVENTURE - AFTER

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# INTRODUCTION

- Let’s start by a question: What is testing in your opinion ?

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# INTRODUCTION

- Let’s start by a question: What is testing in your opinion ?

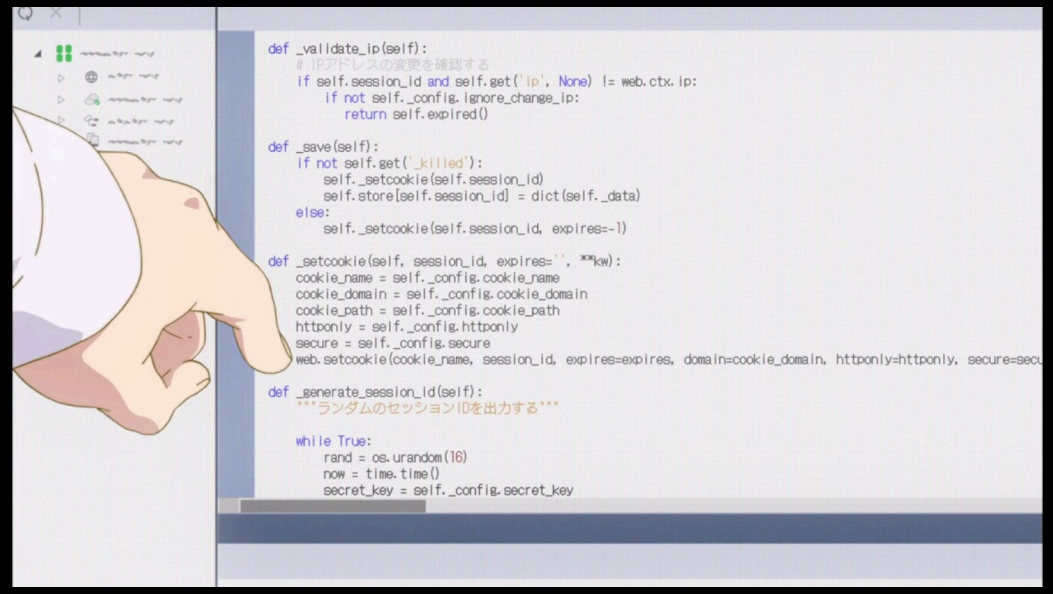
Testing is not debugging. But debugging is a part of testing.

- Testing is a process in which execution is controlled in order to answer ONE question: « Does the software **behave** as **specified** ? »

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# INTRODUCTION

- What do you need ?



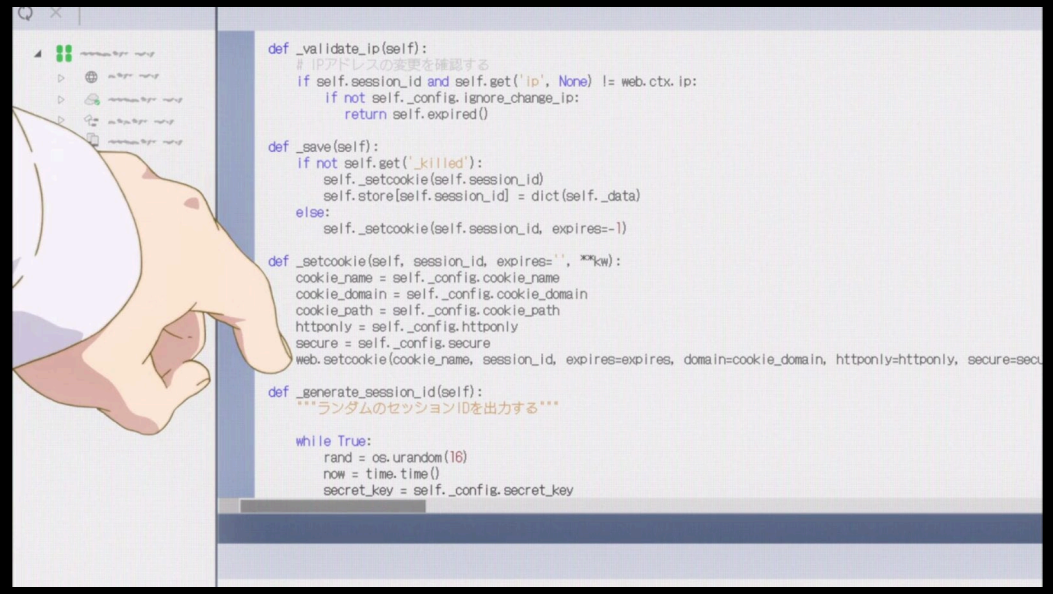

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# INTRODUCTION

- What do you need ?
  - A specification (correctness) :

  « If the user clicks on the « start » button, the « start menu » appears ».

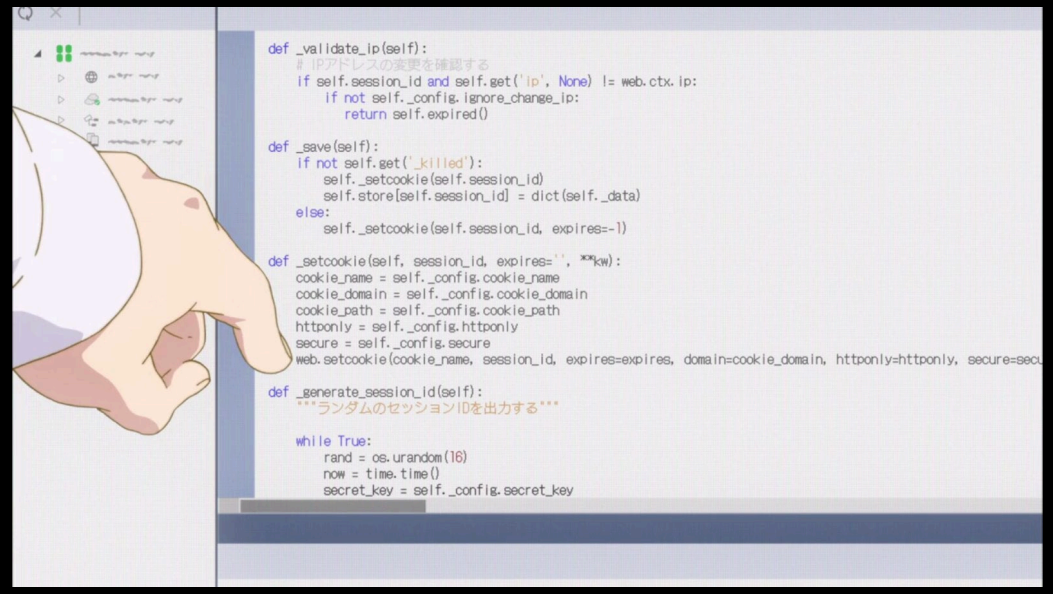



 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# INTRODUCTION

- What do you need ?
  - Or a *property* :
    - No dead code (all paths are reachable)
    - Safety : nothing bad happens => Your software always stays within some allowed region
    - Liveness : Your software always progresses.

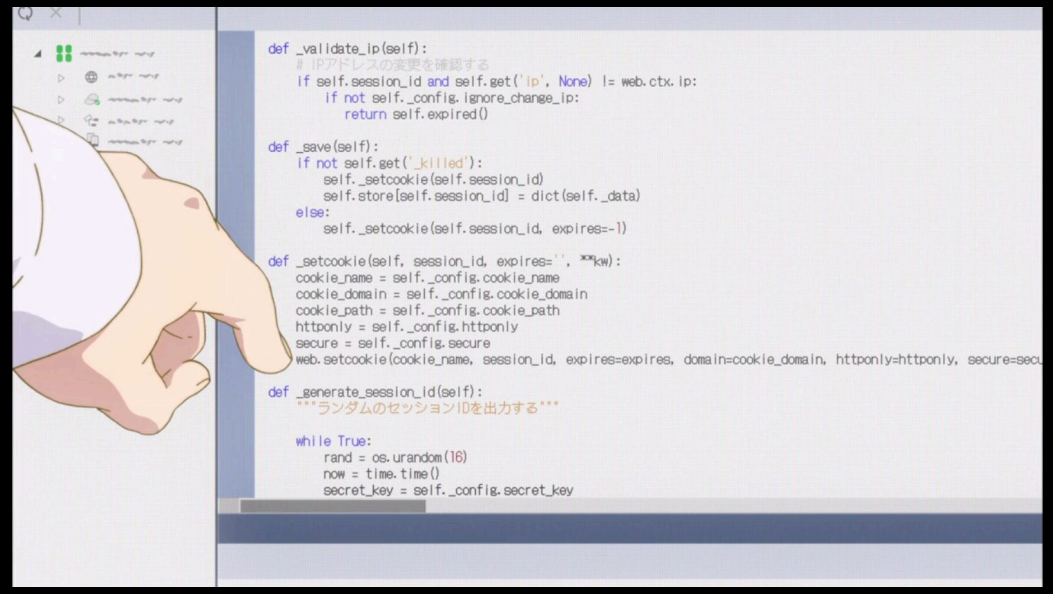



 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# INTRODUCTION

- What do you need ?
  - A specification / property
  - A working program / Piece of Working Software

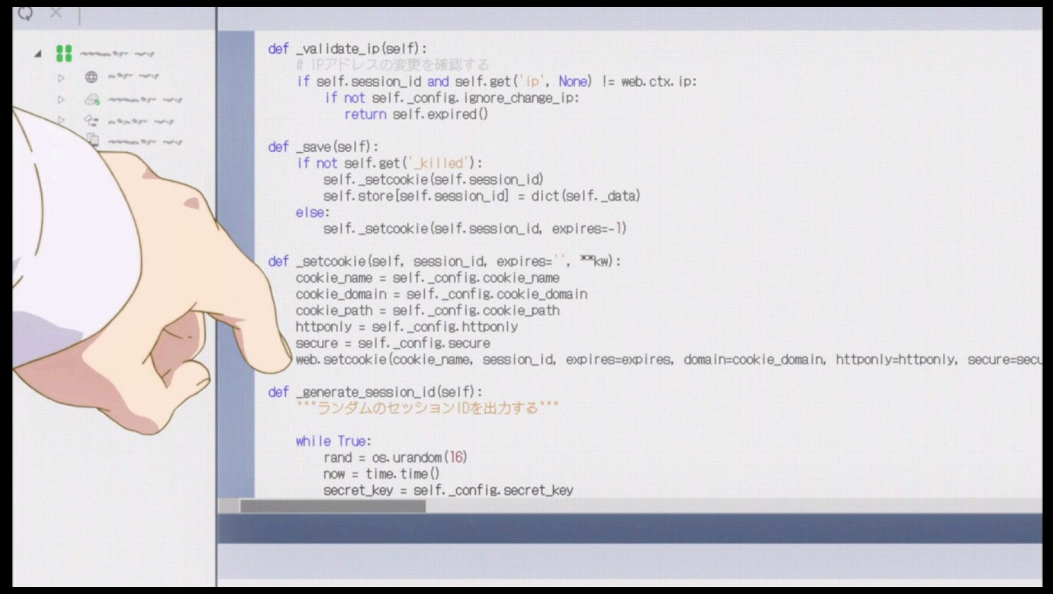



 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# INTRODUCTION

- What do you need ?
  - A specification / property
  - A working program / Piece of Working Software
  - A test plan and methodology




 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# INTRODUCTION

- Testing is a technique involved in two activities:
  - Verification = « Are we doing the job right ? » => *Are we building the software right ?*
  - Checks the conformance and consistency with a given specification

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# INTRODUCTION

- Testing is a technique involved in two activities:
  - Validation = « Are we doing the right job ? » => *Are we building the right software ?*
  - Checks that what has been specified is what the user wanted

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# INTRODUCTION

- WARNING : Testing is not the panace. It’s only a part of the analysis. Inspection and measurement are also required.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# THE TWO PROBLEMS

- You cannot test something you have not thought of.
  - To prepare a test, you need specifications.

- If all the tests pass, it doesn’t mean your software is error-free.
  - Misconception of what a test is.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# DEBUGGING OR TESTING ?

- Remind the terminology :
  - Error = Mistake made by an engineer (misunderstanding a requirement or a design specification)
  - Fault = Manifestation of this error (unexpected behaviour)
  - Failure = Incorrect behaviour that is caused by executing a fault (crash, unrecoverable behaviour)

  - Testing => Discover **failures** !

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

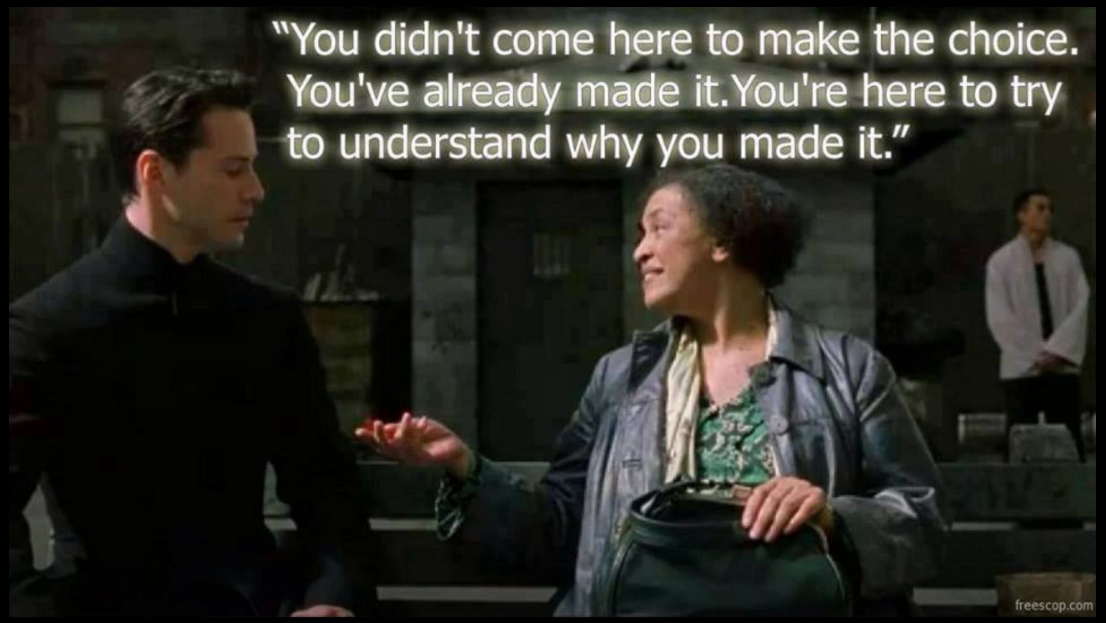
# DEBUGGING OR TESTING ?

- Remind the terminology :
  - A bug / glitch = Error but discovered only through faults, which are discovered only thanks to failures.
  - Debugging = associate failures with faults => process of analyzing and locating bugs when the software does not behave as expected.
  - Testing = process of methodically searching for and exposing bugs (not just fixing those that happen to show up)

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# DEBUGGING OR TESTING ?

Debugging




Testing

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# DEBUGGING OR TESTING ?

- Debugging supports testing.
- If you do debug without understanding => you may add errors.
- If you don’t see any failure, it only means there is no unintended behaviour, but it doesn’t mean there is no error.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# DEBUGGING OR TESTING ?

- There is no possible way to guarantee that you'll find all bugs in a **piece of software**.
- Faults may be hidden in rarely executed code.
- Sometimes, faults are catched, trapped and corrected by another unintended behaviours => bugs neutralization.
- Hidden bugs => Really dangerous because they may cause security issues.

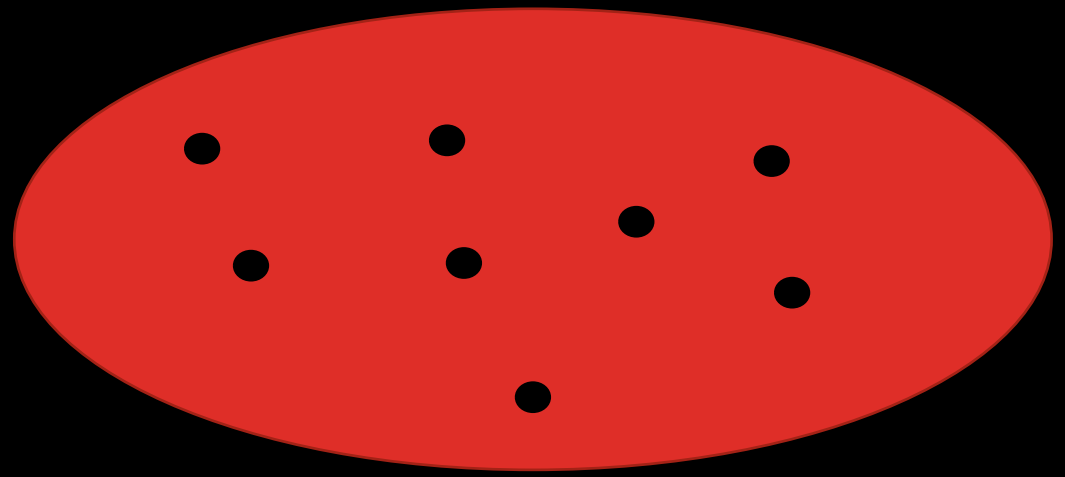
 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# ANOTHER METAPHOR

This is a software with all possible states/behaviours.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# ANOTHER METAPHOR



This is a tested software with 8 discovered failures.

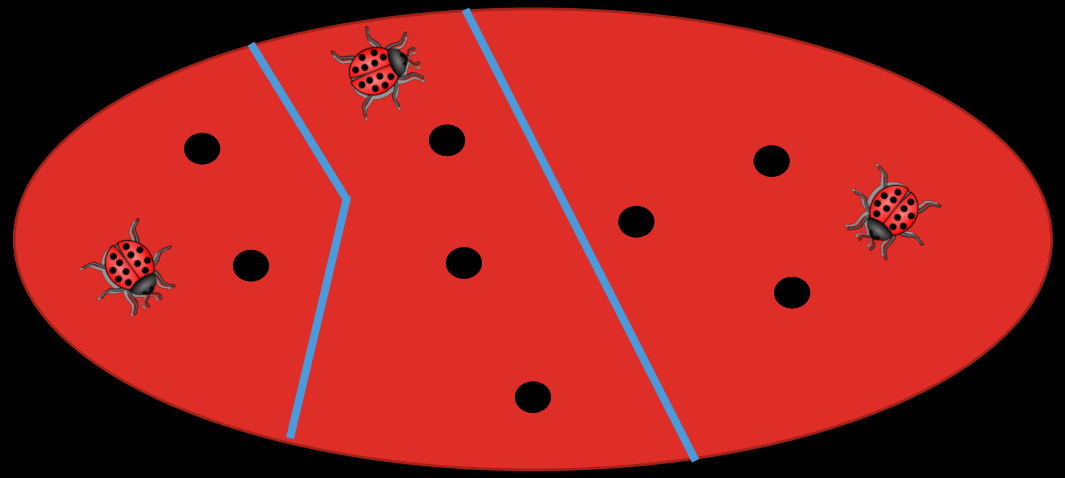
Not really comprehensive ?

Be aware : you discovered 8 failures, but maybe you have much more faults !

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# ANOTHER METAPHOR

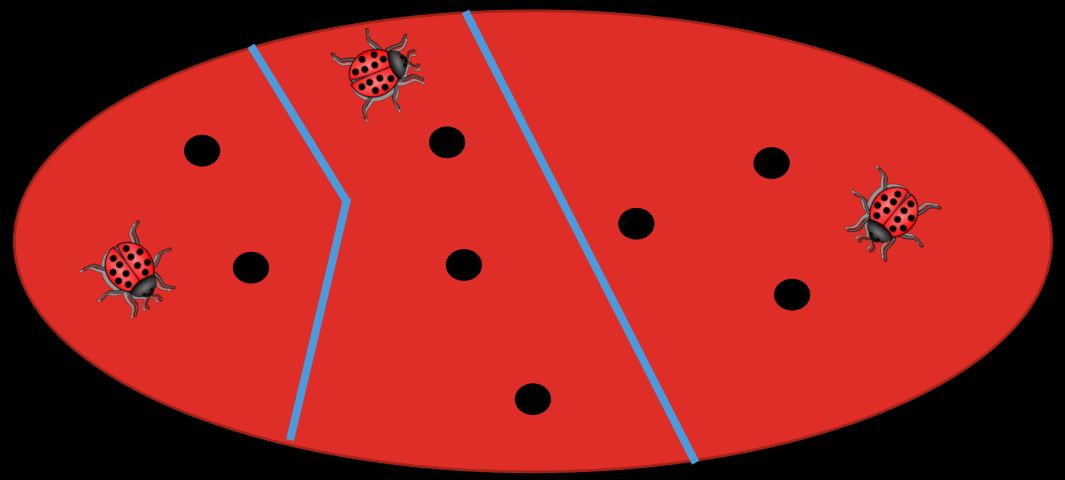


You found the 3 bugs responsible of these 8 failures at least.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# ANOTHER METAPHOR



Indeed, you see only the surface. What is happening inside the cheese ?

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# SOFTWARE TESTING – IMPORTANT KEYS

- Software testing is a set of processes to make sure that:
  - Computer code does what it was designed to do
  - Computer code does not do anything *unintended*.
  - Give inputs to the software and check the outputs.

- Software should be predictable, consistent, no surprising.

- Software Testing cannot be exhaustive, you won't be able to find all the errors.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# SOFTWARE TESTING – IMPORTANT KEYS

- Keep in mind : Software Testing and Program Testing (Proving) is two different things.

- You can completely and exhaustively test a program, but not a piece of software.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# LET'S TAKE ANOTHER EXAMPLE

```
Scanner keyboard = new Scanner(System.in);
int a, b, c;
a = keyboard.nextInt();
b = keyboard.nextInt();
c = keyboard.nextInt();

if (a == b && a == c)
        System.out.println("Equilateral");
else if (a == b || a == c || b == c)
        System.out.println("Isosceles");
else
        System.out.println("Scalene");
```

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# LET'S TAKE ANOTHER EXAMPLE

```
Scanner keyboard = new Scanner(System.in);
int a, b, c;
a = keyboard.nextInt();
b = keyboard.nextInt();
c = keyboard.nextInt();

if (a == b && a == c)
        System.out.println("Equilateral");
else if (a == b || a == c || b == c)
        System.out.println("Isosceles");
else
        System.out.println("Scalene");
```

- There is an infinite number of test cases for the software
- You can fix some values for a,b,c and tests

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# LET'S TAKE ANOTHER EXAMPLE

```
Scanner keyboard = new Scanner(System.in);
int a, b, c;
a = keyboard.nextInt();
b = keyboard.nextInt();
c = keyboard.nextInt();

if (a == b && a == c)
        System.out.println("Equilateral");
else if (a == b || a == c || b == c)
        System.out.println("Isosceles");
else
        System.out.println("Scalene");
```

- a = 1
- b = 1
- c = 1

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# LET'S TAKE ANOTHER EXAMPLE

```
Scanner keyboard = new Scanner(System.in);
int a, b, c;
a = keyboard.nextInt();
b = keyboard.nextInt();
c = keyboard.nextInt();

if (a == b && a == c)
        System.out.println("Equilateral");
else if (a == b || a == c || b == c)
        System.out.println("Isosceles");
else
        System.out.println("Scalene");
```

- a = 3
- b = 1
- c = 5

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# LET'S TAKE ANOTHER EXAMPLE

```
Scanner keyboard = new Scanner(System.in);
int a, b, c;
a = keyboard.nextInt();
b = keyboard.nextInt();
c = keyboard.nextInt();

if (a == b && a == c)
        System.out.println("Equilateral");
else if (a == b || a == c || b == c)
        System.out.println("Isosceles");
else
        System.out.println("Scalene");
```

- a = 2
- b = 2
- c = 8

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# LET'S TAKE ANOTHER EXAMPLE

```
Scanner keyboard = new Scanner(System.in);
int a, b, c;
a = keyboard.nextInt();
b = keyboard.nextInt();
c = keyboard.nextInt();

if (a == b && a == c)
        System.out.println("Equilateral");
else if (a == b || a == c || b == c)
        System.out.println("Isosceles");
else
        System.out.println("Scalene");
```

So do you think this software has bugs ?

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# LET'S TAKE ANOTHER EXAMPLE

```
Scanner keyboard = new Scanner(System.in);
int a, b, c;
a = keyboard.nextInt();
b = keyboard.nextInt();
c = keyboard.nextInt();

if (a == b && a == c)
        System.out.println("Equilateral");
else if (a == b || a == c || b == c)
        System.out.println("Isosceles");
else
        System.out.println("Scalene");
```

- a = -6
- b = 2
- c = 2

This program is correct but is the software really correct ?

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# LET'S TAKE ANOTHER EXAMPLE

```
Scanner keyboard = new Scanner(System.in);
int a, b, c;
a = keyboard.nextInt();
b = keyboard.nextInt();
c = keyboard.nextInt();

if (a == b && a == c)
        System.out.println("Equilateral");
else if (a == b || a == c || b == c)
        System.out.println("Isosceles");
else
        System.out.println("Scalene");
```

So actually, you only proved the correctness of the program, you didn't try the software.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# THIS IS TESTING THE FUNCTION

Test Scenarios

| Test Cases | Test Results |
|---|---|
| Valid scalene triangle i.e. 1, 2, 3 or 2, 5, 10 are not valid triangles | FAIL |
| Valid equilateral triangle | PASS? |
| Valid isosceles triangle (Note: 2, 2, 4 is not a valid triangle) | FAIL |
| Valid permutations for the sides of the isosceles (3,3,4; 3,4,3; 4,3,3) | PASS? |
| Test case with one side a zero value | FAIL |
| Test case with one side being a negative value | FAIL |
| Sum of two positive integers equal to the third = invalid triangle! (1,2,3) | FAIL |
| Test all permutations of the previous case | FAIL |
| Three integers >0 such that the sum of two of the numbers is less than the third | FAIL |
| Test all permutations of the previous case | FAIL |
| Test case in which all sides are zero | FAIL |
| Test for non-integer values | FAIL |
| Test for the wrong number of values (i.e. 2 instead of 3 integers) | FAIL |
| For each test case specify the expected output in addition to the input values | FAIL |

Expected results

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# COMPLETENESS

- So, completely testing a program (underlying code) is possible
  - Set of natural numbers is infinite
  - Set of integers is finite, coverage of all statements, etc.
- But, completely testing a system is impossible
  - Need of heuristics (reduce the state space, fold into different subspaces)
  - Create tests that sample the input/output for each partitions

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# CHALLENGES

- If it’s not possible to find all the possible errors / exhaustively test the software, how increase quality ?
- Testing a trivial program is already hard, what about complex software ?
- Testing consumes a lot of time… how to perform it efficiently ?

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# PSYCHOLOGY AND ECONOMICS

- You have to agree that :
  - Test every possible combination of input/output/behaviour is not possible.
  - Complete testing is not economically feasible (need too much time and resources)
  - Do exactly as usual:
    - Use a methodology
    - Create a lifecycle

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# PSYCHOLOGY AND ECONOMICS

Bad approach :

- « We'll test to demonstrate there is no error ».
- « We'll test to show that our software performs its intended functions correctly ».
- « We'll establish confidence that our software does what it's supposed to do ».

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# PSYCHOLOGY AND ECONOMICS

- Don’t show that your software works !
- Start point: your program contains errors.
- Test to find as many errors as possible:
  - Execute your program with the intent of finding errors => Try everything to make it crash !

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# PSYCHOLOGY AND ECONOMICS

- Other psychological advices:
  - If the software passes a test case successfully, it's a failure.
  - If the software fails to pass a test case, it's a success.

- Unsuccessful test case = success for you
- Successful test case = unsuccess for you

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# PSYCHOLOGY AND ECONOMICS

- Admitting you have a pain. You go to the doctor and he doesn’t find any problems. Would you say these are successful tests ?
- If everything goes too much well, don’t give up the exploration.
- Not finding an error = uncertainty that this error exist.
- Finding an error = certainty that it won’t exist anymore.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# PSYCHOLOGY AND ECONOMICS

- So testing is : « Proving the presence of bugs, not showing their absence ! » (Dijkstra)
- Testing need planning and strategies.
- Depending on the SDLC and the company,
  - Tests may be written other developers than the responsible of the task
  - Tests may be conducted by another team.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# PSYCHOLOGY AND ECONOMICS

- Two tasks:
  - Write the tests ⇔ Think about testing scenarios before programming.
  - Implement the tests ⇔ Test your program **manually** or implement a program which will test your software **automatically**.
- In the second case :
  - Written test = specifications of your automatic test
  - Automatic test = program => need a design and ensure that this testing program is also correct (This is another problem !)

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# TESTS CLASSES

- Two categories of tests:
  - Static tests => these tests work on the code without executing it (we talk about proofs).
  - Dynamic tests => these tests execute the software on a particular subset of inputs.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# SYSTEMATIC TESTING

- Systematic testing is an explicit discipline or procedure (system) for:
  - Choosing and creating test cases
  - Executing test cases and documenting the results
  - Evaluating (automatically) the results
  - Deciding when tests are finished (or if necessary to perform more tests)

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# SYSTEMATIC TESTING

- Each test scenario and methods focus on a particular point of view (test criterion)
- Each test scenario relies on a specific specification
- Everything is testable :
  - Modules
  - Classes
  - Functions
  - Etc.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# LEVELS OF SPECIFICATIONS

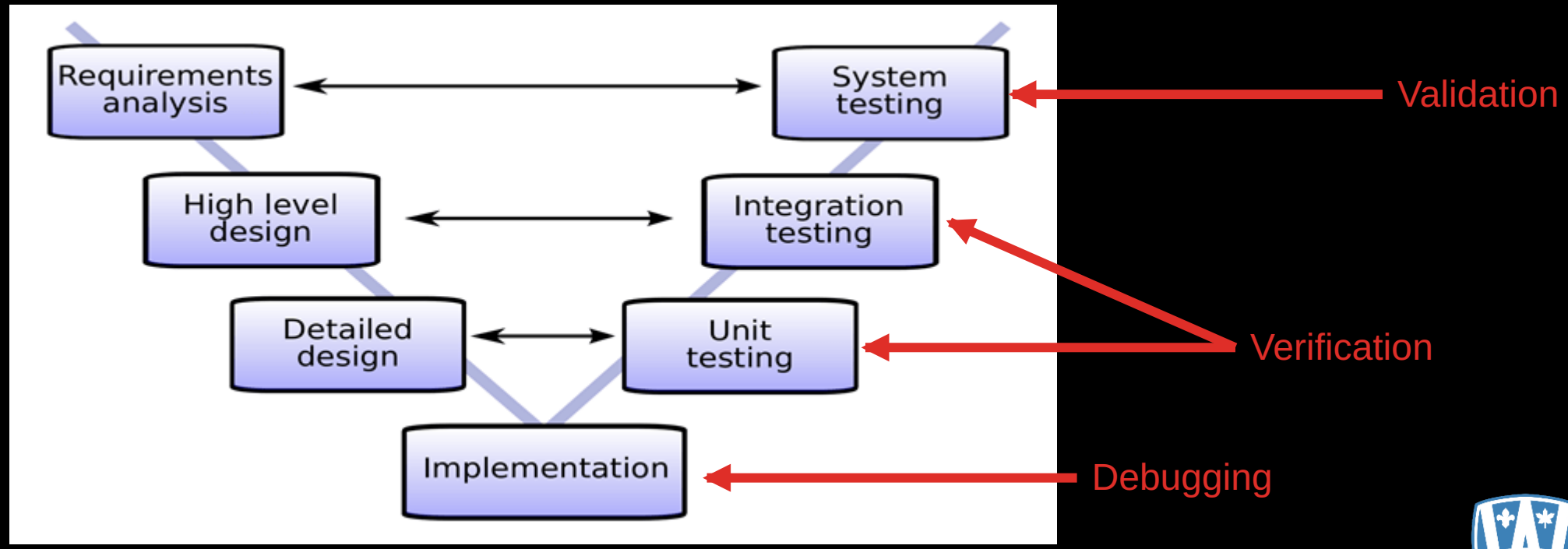
- Reminder:
  - Everything is a model from the specifications to the code
  - A model is a specification of another model

- 3 levers of specifications:
  - Functional specifications => What the software does
  - Design specifications (high-level) => Architecture and components
  - Detailed Design specifications (low-level) => How each component are implemented / Code unit

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# LEVELS OF TESTING

- If you have 3 levels of specifications, you have **at least** 3 levels of testing !




 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# UNIT TESTS

3. Unit testing = Verification of individual components meet their detailed design specification (UML diagrams, pseudo-code, …)

2. Integration testing = Verification that a group of units corresponding to architectural elements of the high-level design can be integrated to work as a whole

1. System testing = Checking that the total product has the functionality specified in the functional specification.

0. Acceptance testing = Validation that the software meets their real intentions and is accepted. Base for Certification.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# INTEGRATION TESTS

- Check that modules work together in combination
- You find delay when you reach this point
  - Hidden assumptions are surfaced when code written by different developers are integrated
- Lack of integration testing has already led to spectacular failures (Mars Polar Lander)

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# EXERCISE

Imagine a car with an engine and wheels. Imagine unit tests and integration tests.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# SYSTEM TESTS

- Test performed by the developer to ensure that all functionalities have been implemented
  - Have all user stories been implemented and works correctly ?

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# ACCEPTANCE TESTS

- Tests performed by the customer or the user to check that the system meets the initial needs

- On large project, developers go on site and install the system, and respond to problems as they arise

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# TEST-DRIVEN DEVELOPMENT

- V-Model

1. Write one level of specification.
2. Write the corresponding tests.
3. Design test without knowledge of the implementation.

- TDD-Model

1. Tests is the specifications.
2. Implement the test right before the implementation.

Test what the software ACTUALLY does, NOT what it should do.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# TEST EVALUATION

- Within each level of testing, evaluate results:
  - If a test fails, check the test and apply it again (maybe the test is wrong)
  - If a test fails, fix the software and apply it again (maybe the software is wrong)
- Apply tests until no more problems are found at this level.
- Proceed with the next level of testing.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# TEST EVOLUTION

- Tests never die, even when the software enters production.
- Repeat, modify, extend your tests. Each time you implement a new functionality, execute the previous tests to check if no other functionalities are broken.
- Maintenance of tests themselves is also a part of maintaining a software.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# CONTINUOUS TESTING

- Testing is a continuous process !
- At every stage of the SDLC, you have a task related to tests (including in Agile methods)
  - Example: During requirement gathering, ask your customer if you get it.
- Iterative SDLC => Emphasis at the end of each iteration

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# CONTINUOUS TESTING

- Automate your tests !
  - You will be able to run them as soon as possible
  - You will be able to run them as changes are made
- Tests can take longer time when your software grows
  - Execute short tests multiple times per day
  - Execute long tests at least once per day
- Use testing framework

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# CONTINUOUS INTEGRATION

- A CI tool wrap:
  - Version control
  - Compilation
  - Testing
- Principle : as soon as you version your code, the repository is automatically compiled and tests are automatically performed. The system reports back to you if something went wrong with the integration.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# CONTINUOUS INTEGRATION

- Example of CI system:
  - Jenkins
  - Hudson
  - GitLab
  - Helix
  - Atlassian JIRA

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# SUMMARY

- Objective of testing : Showing that a program/software has errors.
- Testing is not just bug fixing.
- Testing is based on specifications.
- Testing is done using methodologies and frameworks.
- Testing is a continuous infinite process.

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

# SOFTWARE TESTING VITAL GUIDELINES

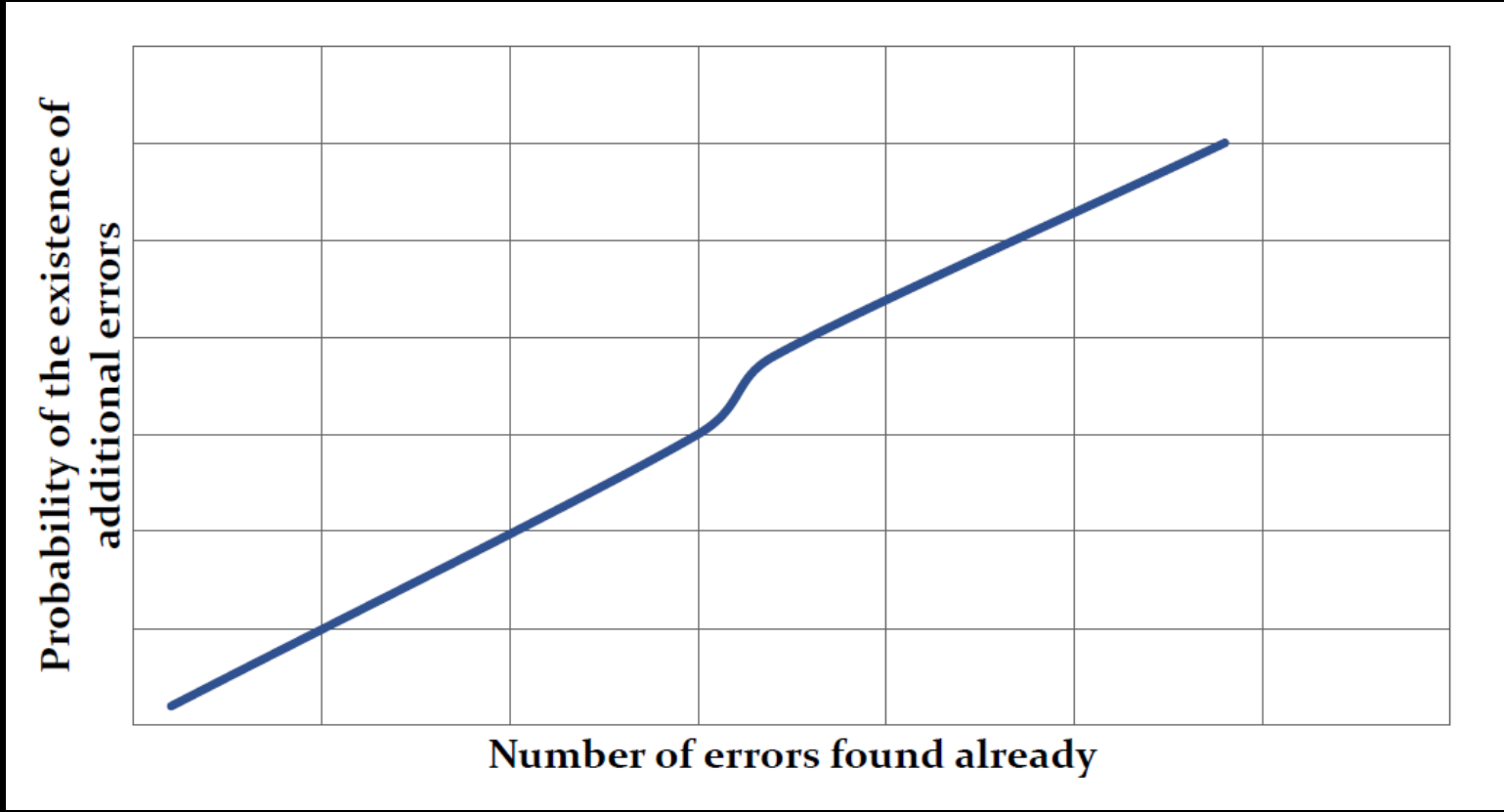
1. A necessary part of a test case is a definition of the expected output or result.
2. A programmer should avoid attempting to test his or her own program.
3. A programming organization should not test its own programs.
4. Thoroughly inspect the results of each test.
5. Test cases must be written for input conditions that are invalid and unexpected, as well as for those that are valid and expected.
6. Examining a program to see if it does not do what it is supposed to do is only half the battle; the other half is seeing whether the program does what it is not supposed to do.
7. Avoid throwaway test cases unless the program is truly a throwaway program.
8. Do not plan a testing effort under the tacit assumption that no errors will be found.
9. The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section.
10. Testing is an extremely creative and intellectually challenging task.

From Ziad Kobti's Lecture (Software Testing), 2012

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# ERRORS REMAINING / ERRORS FOUND




From Ziad Kobti's Lecture (Software Testing), 2012

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor

# REFERENCES

This lecture is based on:

- Introduction to Software Testing – Ziad Kobti – University of Windsor
- Test – Amine Hamri – Aix-Marseille University
- CSCI3060U – Jeremy Bradbury - Ontario Tech University
- CSCI 5828 – Kenneth Anderson
- The Art of Software Testing – Glenford Myers

 Copyright © Dr. Aznam Yacoub 2021 – All rights reserved.

University of Windsor