CMPUT 401 Software Process and Product Management

Ildar Akhmetov

ildar@ualberta.ca

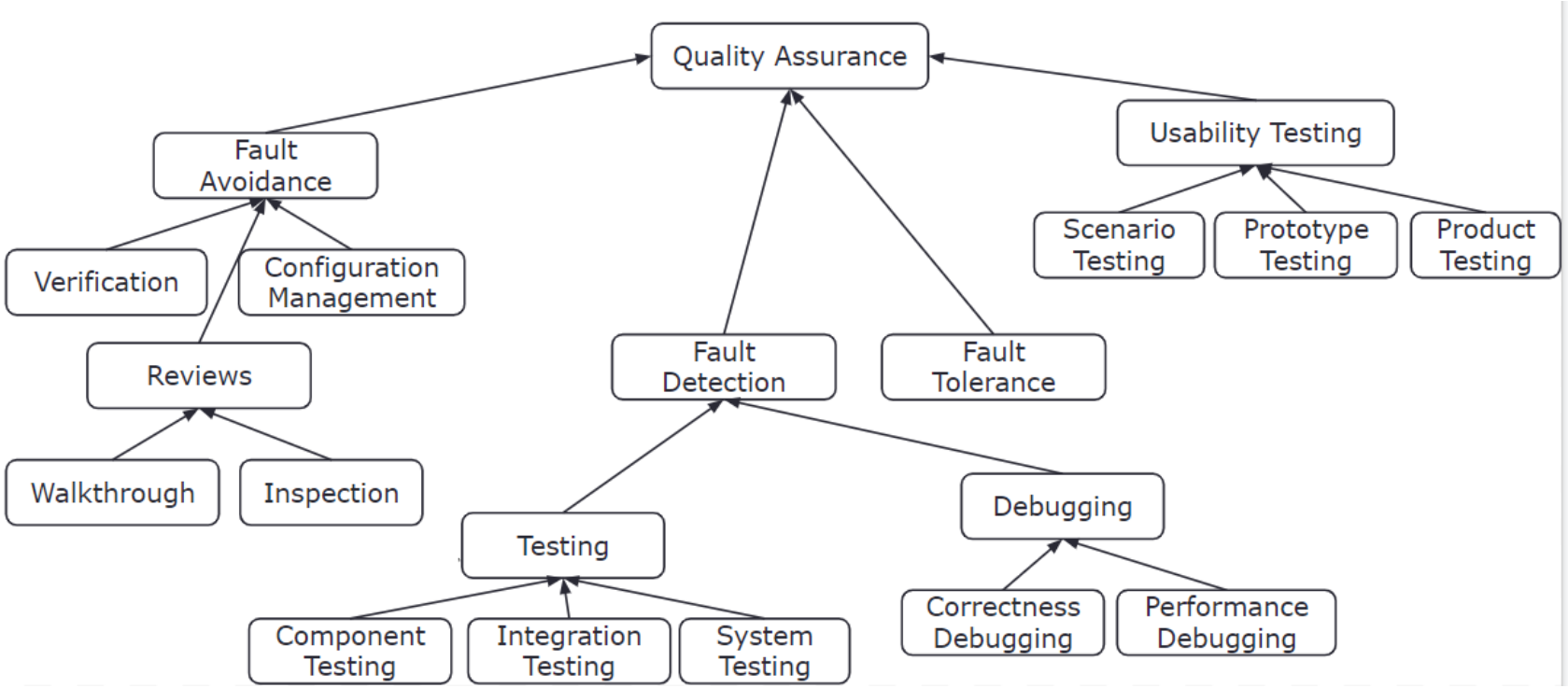
Department of Computing Science

University of Alberta

Testing and Testing Tools

Fall 2020

Quality Assurance



Terminology

 Reliability	how well the system's observed behaviour conforms to specified expected behaviour.
 Failure	a deviation of the observed behaviour from the specified behaviour.
 Error	a state such that further processing by the system will lead to failure.
 Fault (Bug)	the cause of an error.

Finding Faults Early Is Good

		Time Detected				
		Requirements	Architecture	Construction	System Test	Post-Release
Time Introduced	Requirements	1×	3×	5–10×	10×	10–100×
	Architecture	-	1×	10×	15×	25–100×
	Construction	-	-	1×	10×	10–25×

Testing

What is it NOT?

- A "proof of correctness"
- Testing can never completely identify all the defects within software!

What is it?

- A process of validating and verifying that a software program:
  - meets the business and technical requirements
  - works as expected.

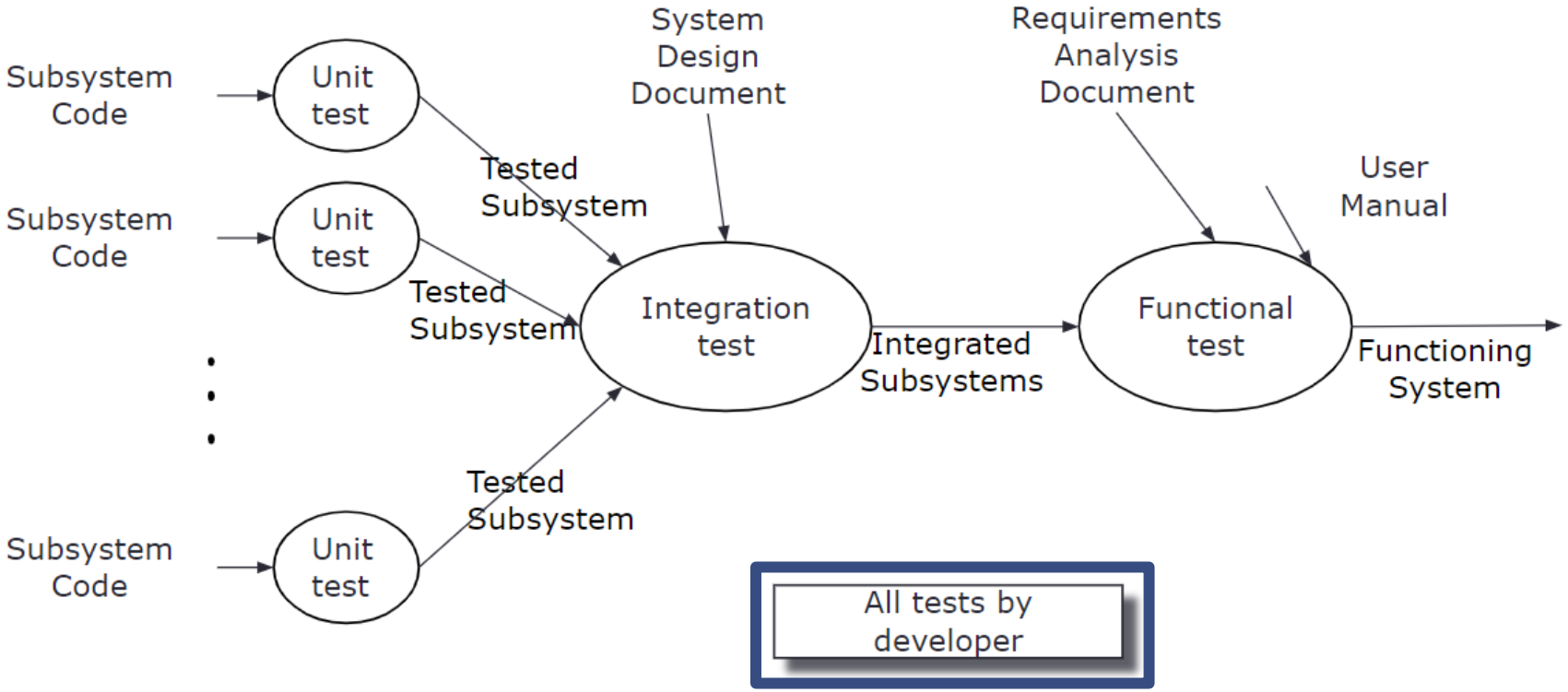
Testing: What is it?

Process of executing a program with the intent of finding errors.

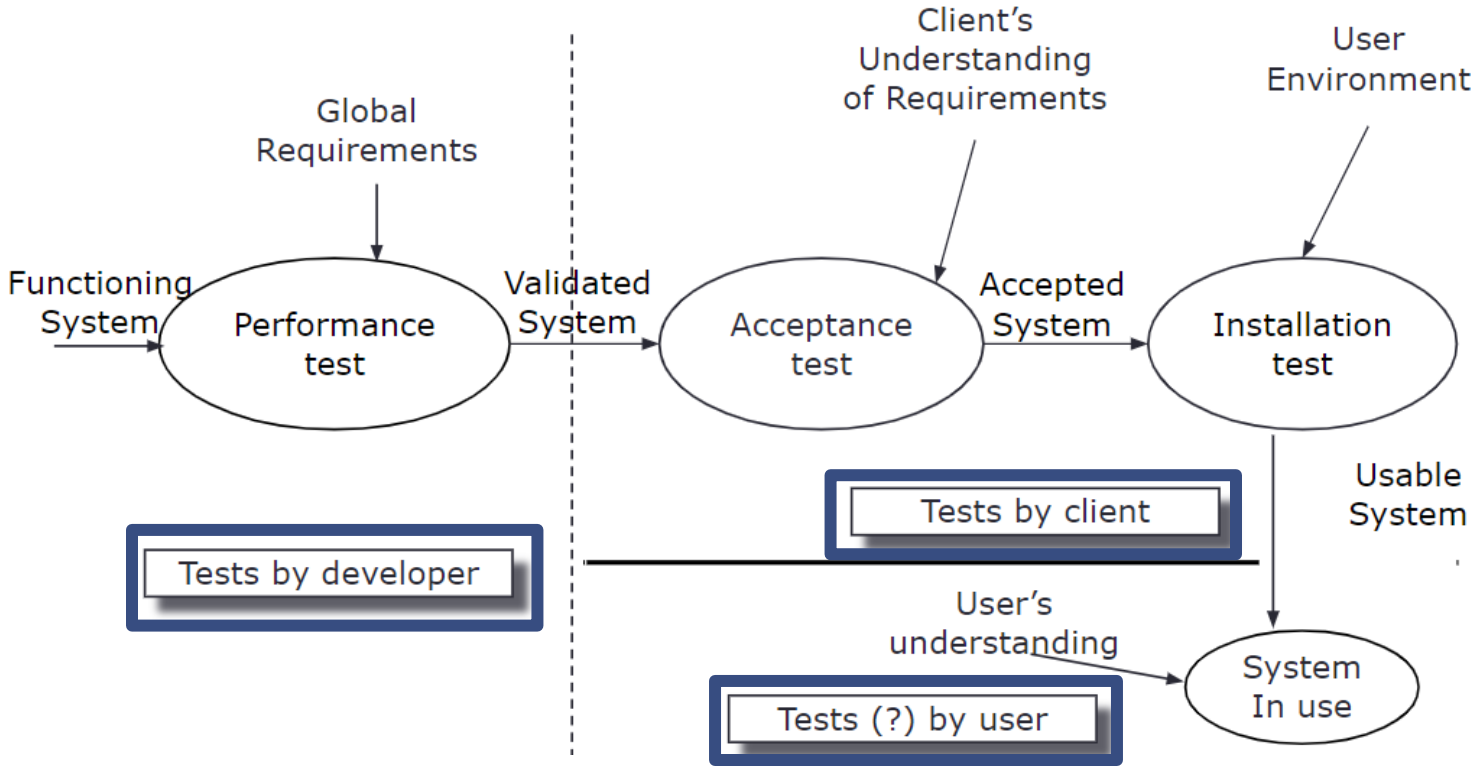
Glenford Myers [The Art of Software Testing, 1979]

- This makes it a challenging task!
  - It is not easy to find errors in software programs (especially, large!)
  - It is a destructive activity - your purpose is to find faults
  - This can be demoralizing and unrewarding if not treated positively

Testing Activities



Testing Activities (cont.)



Five Dimensions of Testing

Testers

- Who does the testing

Coverage

- What gets tested

Potential problems

- Why you're testing

Activities

- How you test

Evaluation

- How to tell whether the test passed of failed

People-based Techniques

User testing	Testing with people similar to potential users. Can be done at any time during development
Alpha testing	In-house testing performed by the test team (and friendly insiders)
Beta testing	Testing by your product's target market when the product is close to completion
Bug bashes	In-house testing using anyone who is available
Subject-matter expert testing	Testing by an expert
Paired testing	Two testers work together to find bugs
Eat your own dogfood	Using prerelease versions of own software

Who Are the Testers?

Developers are too close to the code and "understand" it too well

- Often driven by delivery schedules and are busy fixing found bugs

Independent testers will try to break the code and are by quality concerns

- Disadvantage: it will take them some additional time to learn the system

Testing Takes Creativity!

- Testing is often viewed as dirty work!
- To develop an effective test, one must have
  - Detailed understanding of the system
  - Knowledge of the testing techniques
  - Skill to apply these techniques effectively and efficiently
- Programmers often stick to the data set that makes the program work
- A program often does not work when tried by somebody else
  - Don't let this be the end-user!

Coverage-based Techniques (1)

Function testing	Test every function, one by one
Feature integration testing	Test multiple functions together
Menu tour	Go through all menus and dialogues
Equivalence class analysis	Find an equivalence class and test only one or two of its members
Boundary testing	Use only boundary values of each equivalence class
Input field test matrix	For each input field type, develop a set of test cases

Coverage-based Techniques (2)

Logic testing	Check every logical relationship in the program
State-based testing	Test in each state
Path testing	Path = all steps user took to get to the current state. Determine different paths and test them
Specification-based testing	Testing each specification claim (including manual, marketing docs, ads)
Requirements-based testing	Testing that each requirement is satisfied

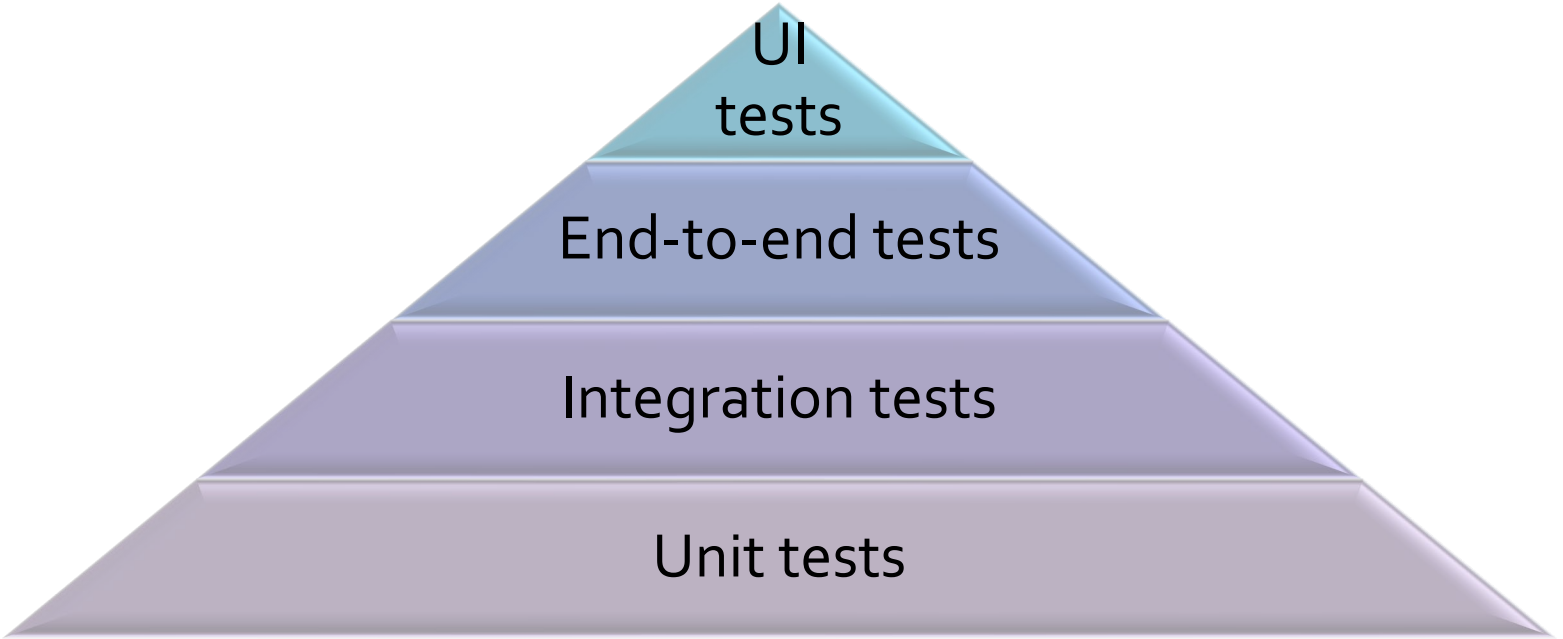
Activity-based Techniques (1)

Regression testing	Reuse of the same tests (e.g., **bug fix regression** = retesting after fixing a bug)
Scripted testing	Manual testing using a step-by-step procedure
Smoke testing	Testing just things that expected to work (to prove that a new build is not worth testing)
Guerilla testing	A fast and vicious testing, time-boxed and done by an experienced exploratory tester
Scenario testing	Derived from use cases 1) **Realistic** (reflects something that the users would actually do) 2) **Complex** (should be challenging to the program) 3) **Easy to tell** whether the program passed or failed the test

Activity-based Techniques (2)

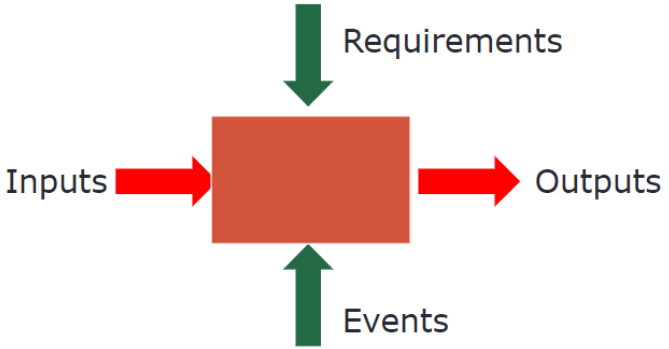
Installation testing	Installing software in various ways, on various types of systems
Load testing	Attacking the system by a high enough load
Reliability testing	Testing the system for several days or weeks
Performance testing	Determining how quickly the program runs; does it need optimization

Testing Hierarchy



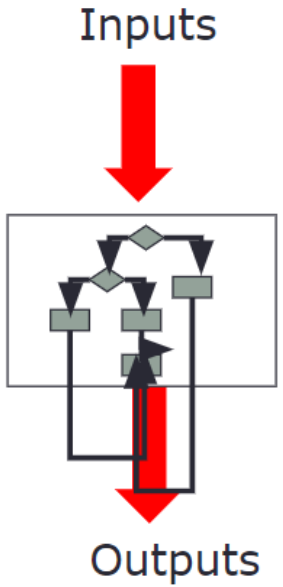
Types of Unit Testing

Black Box testing (functional, specification-based testing) inspects the module from the outside



Types of Unit Testing

White Box testing
(logical, structural, or program-based testing) looks "under the cover"



Black Box Testing: What Can Be Discovered?

- Parameter (interface) errors
- Missing or misbehaving functions
- Errors in data structures or databases
- Erroneous initialization and termination conditions
- Performance or security errors

Black Box Testing Example

- Conditions for the Bill Gates' Scholarship
- Find a set of **boundary value** test cases

GPA > 3.7
3rd **year** of **CSc Program**
Your parent's **total income** < $100K.
IF a student qualifies, return "Yes";
ELSE return "No".

Example: Test Cases

Boundary values, Equivalence classes

GPA	Program	Year	Income		Result
3.8	CSc	3	100		No
3.7	CSc	3	100		No
3.8	Bio	3	100		No
3.8	CSc	2	100		No
3.8	CSc	3	99		Yes
0	CSc	3	100		No - Error Invalid GPA
-1	CSc	3	100		No - Error Invalid GPA
4.1	CSc	3	100		No - Error Invalid GPA
3.8	CSc	0	100		No - Error Invalid Program/Year
3.8	CSc	-1	100		No - Error Invalid Program/Year
3.8	CSc	3	-1		No - Error Invalid Parent Income

Test Matrix

tests (equivalence classes, could be "scripted")

situations (input fields)

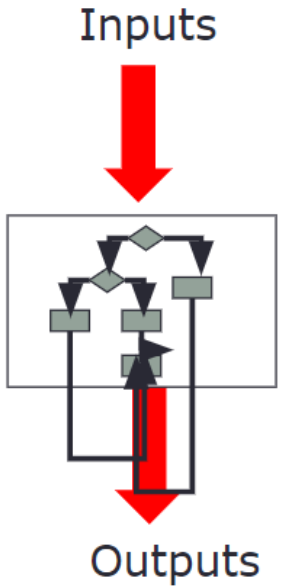
Values / Fields	Nothing	Empty	0	LB–1	LB	UB	UB+1	...	_	O	modifier	fkey	...
Name													
Address													
Amount													

Traceability Matrix

tests	items (features, functions, ...)					
	S1	**S2**	**S3**	**S4**	**S5**	**S6**
T1	x		x			x
T2	x	x		x		x
T3			x	x		x
T4			x	x		x
T5	x				x	x
T6		x				x
	3	2	3	3	1	6

White Box Testing

- White Box - logical testing => look "under the cover"
- Even more difficult and time consuming then black-box testing
- Less popular than black box
- Primarily used in testing key methods/modules



White Box Testing Objectives

Basis path testing	Exercise all independent paths within a module at least once
Condition testing	Exercise all logical decisions on their true and false sides
Loop testing	Exercise all loops at their boundaries and within their operational bounds
Dataflow testing	Exercise all internal data structures to assure their validity

HOW GOOD ARE YOUR TESTS?

Mutation

Bebugging

Mutation Testing

- Make a small change to the program
- The effect of the change should show up in some test!
- If a mutant was introduced without any change in the product behavior:
  - either the mutated code was dead code;
  - or the test suite was insufficient.

Image credit: https://www.nytimes.com/2012/09/30/arts/television/teenage-mutant-ninja-turtles-retooled-for-nickelodeon.html

Bebugging

- Introduce representative bugs to check the effectiveness of the testers
- Known bugs are randomly added to a program source code and the programmer is tasked to find them.
- The percentage of the known bugs not found gives an indication of the real bugs that remain!

Image credit: https://www.jcehrlich.com/blog/cartoon-insects/

Summary: Suggested Practices

- Use unit testing, with "100% coverage" for important modules
- Use integration testing
- Use end-to-end system testing
- Use automated regression tests if possible
- Test from the user's point of view (verify each requirement is met; check for usability)
- Subject the software to stress (high load, limited resources, maxed users)
- Beta test with representative users

References

- Kaner, C., Bach, J., & Pettichord, B. (2002). *Lessons learned in software testing: A context-driven approach*. New York: Wiley. Retrieved from https://learning.oreilly.com/library/view/lessons-learned-in/9780471081128/
- http://pages.cpsc.ucalgary.ca/~eberly/Courses/CPSC333/Lectures/Testing/intro.html
- http://www.rspa.com/spi/test-methods.html
- https://students.cs.byu.edu/~cs340ta/fall2018/readings/WhiteBox.pdf