

Testing - I

Outline

Testing will be covered during TWO lectures.

What is Testing?

Limits of Testing

Testing Strategies

Testing Tactics

Topics covered in second lecture:

Example

Doing the Testing

Testing Testing

Process Issues

Testing

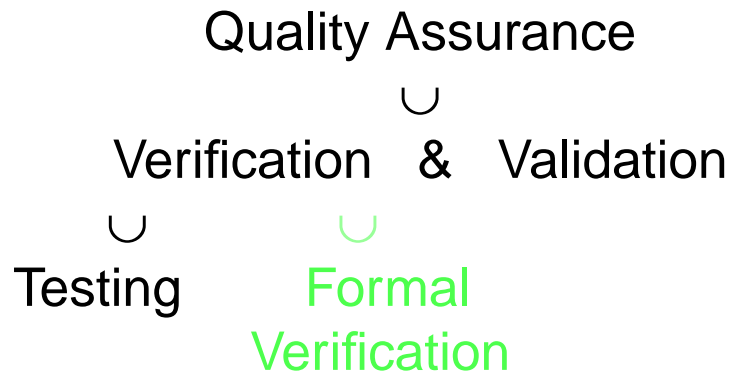
“Testing is a process of inferring certain behavioral properties of a product based on the results of executing the product in a known environment with selected inputs.”

– J. B. Goodenough

Importance of Testing:

- “Checkout” and testing often consume 50% of software effort.
– Barry Boehm
- The cost of fixing an error increases by an order of magnitude at each stage in development.
– Barry Boehm

Where Testing Fits In



Verification:

“Are we building the product right?”

Validation:

“Are we building the right product?”

Testing:

“Are there failures?”

“What are the failures?”

Debugging:

“Why do they occur?”

Goals for Testing (Subset of Goals for QA)

Utility/Correctness

- Meets users needs

- Meets functional specifications

Reliability

- Avoids unacceptable behavior under normal conditions

Robustness

- Behaves properly across a variety of conditions and inputs, normal and abnormal

Availability

- Recovers gracefully and quickly from anomalous behavior or conditions

Maintainability

Efficiency/Performance

Security

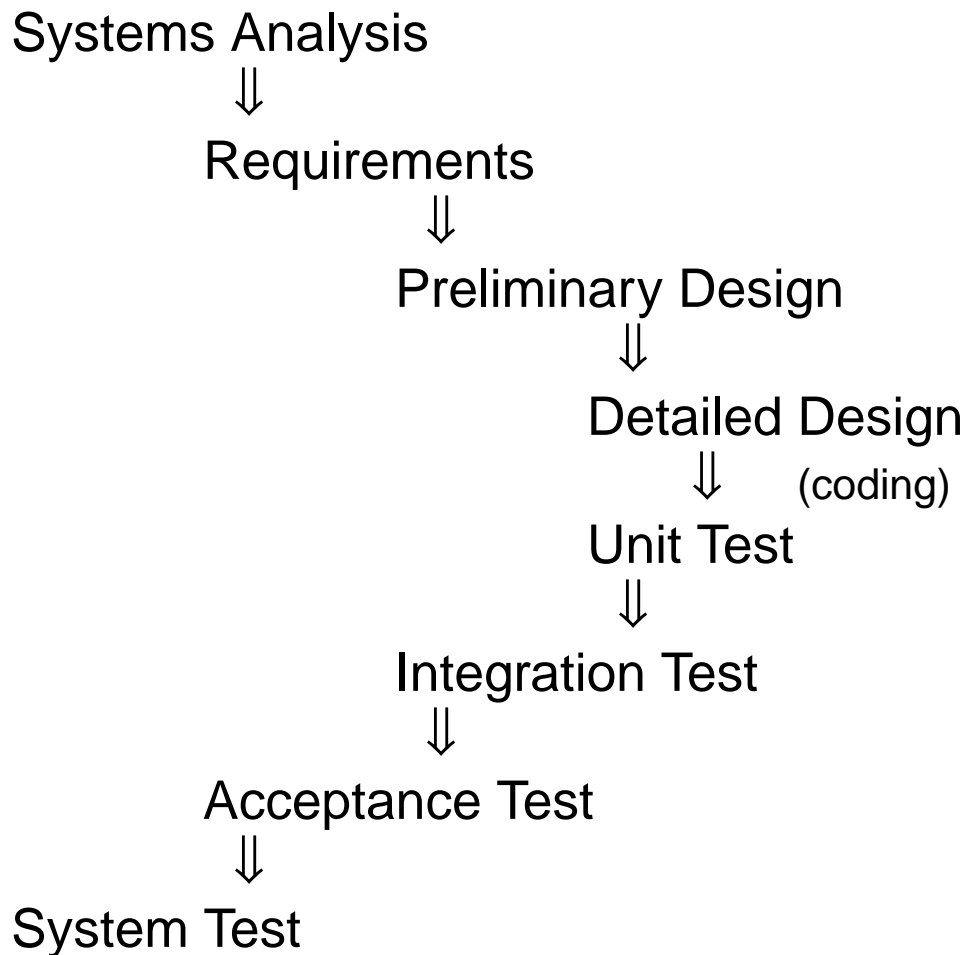
Many of the above seem quite similar, but that's why it's important to recognize all the variations

Limits on Testing

- Can't test-in quality
- Program testing can be used to show the *presence* of bugs, but never their absence.
 - Edsger Dijkstra
- Combinatorial explosion
Exhaustive testing isn't possible in most cases
- Concurrency makes it worse!
Difficult to anticipate concurrent behavior and to control testing in concurrent environment
- Theoretical limitations

Testing's Place in Process

- Test cycle mirrors life cycle:



- Integration & testing happen inside to out

Integration Testing Strategies

Test Strategy \rightleftarrows Integration Strategy

Top-down

entire layer

module at-a-time

Bottom-up

Big-bang

Vertical slice

The following are less integration-dependent:

“Sandwich” testing

Testing by ADT

way to do controlled bottom-up testing in otherwise top-down context

Testing by (global) dataflow

I/O, traversal of program tree

Other Strategy Issues

Unit testing phase

Regression testing

retest units after changes in them or related units
part of CCCM

System testing phase

Critical systems require comparison testing

two or more completely independent (redundant) systems
are developed and the results compared

Process & Methodology

Appropriate tools and techniques

Organization and planning vital

Testing testing

Tests *must* be evaluated for adequacy

Tactical Issues for Testing

Driven by:

- Inputs
- Output
- Internals

Test Set vs. Test Execution

Black Box Testing

- Ignores internals
- Driven by specifications
- Suitable for separate V&V team
- Unit testing up to entire system

Glass Box Testing

- Recognizes internals
(i.e. uses source code)
- Driven by design and implementation
- Unit testing
- Can apply tools
- Suitable for test evaluation

Black Box Testing Techniques

- Techniques for test set definition
- ★ Driven by specifications

Equivalence Partitioning:

- specifications divide input into *valid* and *invalid* classes
- class usually a *range* or *set* of values
- if complex conditions (e.g. regular expressions), treat classes as intersection or union of simple cases
- tests should cover classes independently

Boundary value analysis (for *range* classes):

- ask: “What causes [or changes] condition?”
“What is dividing line?” “Why?”
- test just above and just below range endpoints
- treat number-of-items like a range
- attempt to generate *output* at boundaries too

Black Box Testing Techniques

Heuristics for menus and commands:

- ◇ decide which fields interact
- try both blatant and subtle incorrect forms
- try input that is correct except for order
- try partial correct commands
- try null entries
- try over-constrained commands or too much qualifying data
- try interchanged, mismatched, and wrong delimiters
- try interrupts, reset, other unusual timing problems

Black Box Testing Techniques

Heuristics for forms:

- check functionality as above
- ★ check format and appearance too!
- try all form modes
- try all button modes
visible/invisible, active/inactive, *etc.*
- try help for every item
- functionality of UI platform typically beyond scope
- * usability tests done at unit test time give better chance for correction

Glass Box Testing

☆ Driven by design and implementation

Techniques

Develop appropriate **sets** of test inputs in order to *cover* some aspect of a module

- Condition testing
- Node testing
- Edge testing
- Path testing
- Dataflow testing

Test–Set Evaluation

- Path/condition/flow coverage
- Mutation testing

☆ Indicates where to test, not how to test

e.g. GB \Rightarrow test both branches of `if (X<0)`

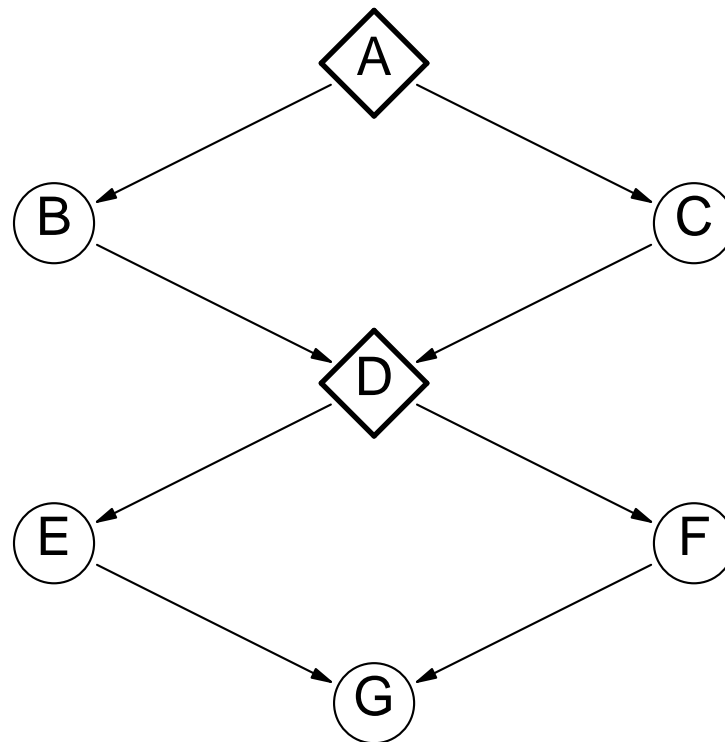
context	instance definition
X is input	trivial/automatic
X is computed	test designer
x<0 is exception	???

Condition Testing

- **cover** logical conditions, such as in *if* statements
- each condition must be evaluated yielding *TRUE* and *FALSE* if “significant” test
- treat “ $A > 0$ AND $B > C$ ” as *two* conditions
 - result of “ $B > C$ ” irrelevant if “ $A > 0$ ” is *FALSE*
 - hence, other conditions in AND must be *TRUE*
 - similarly, other conditions in OR must be *FALSE*

Path Testing

- really tests edges & paths
- **cover** paths in flow chart
- all possible path combinations too numerous to test
- hence, test “independent basis set”



- possible basis for above
 - A - B - D - E - G
 - A - C - D - F - G
- some would suggest adding
 - A - B - D - F - Gbecause this has B | C choice independent of E | F choice

Dataflow Testing

- **cover** each possible assignment to use dependency for all variables

Definition: a *dataflow dependency* on variable *A* exists between the first and last statements of

$A \leftarrow \dots$ *assignment*
• • •

$B \leftarrow A + \dots$ *use*

provided no assignment to *A* intervenes

Method:

1. make a graph with program statements as nodes and dataflow as edges
2. develop tests that provide edge covering of graph

Loop Testing

- treat similar to boundary value analysis:

SKIP

ONCE

TWICE

LIMIT-1

LIMIT

LIMIT+1

- nested loops:
 - work inner to outer
 - test each loop as above
 - hold other loops to small number of iterations