



Software Testing

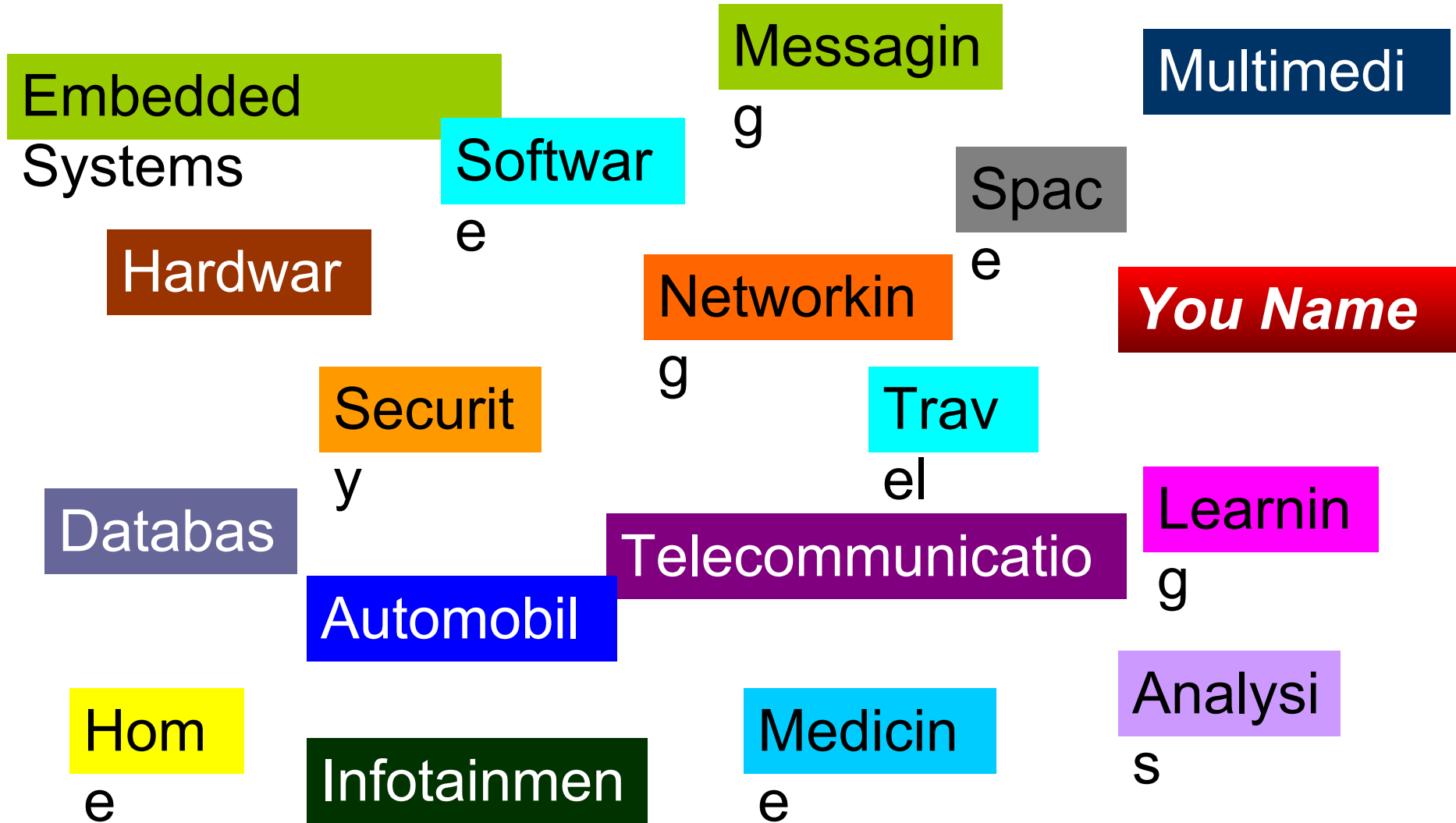


A definition



Testing is a process of
executing a program with
the intent of finding errors

What world are we talking of?



What world are we talking of?

innovate

achieve

lead

Embedded

Systems

Messaging

Multimedia

Software is (nearly)
everywhere. Thus we are
talking about that
“everything”.

Data

Home

entertainment

Infotainment

Medicine

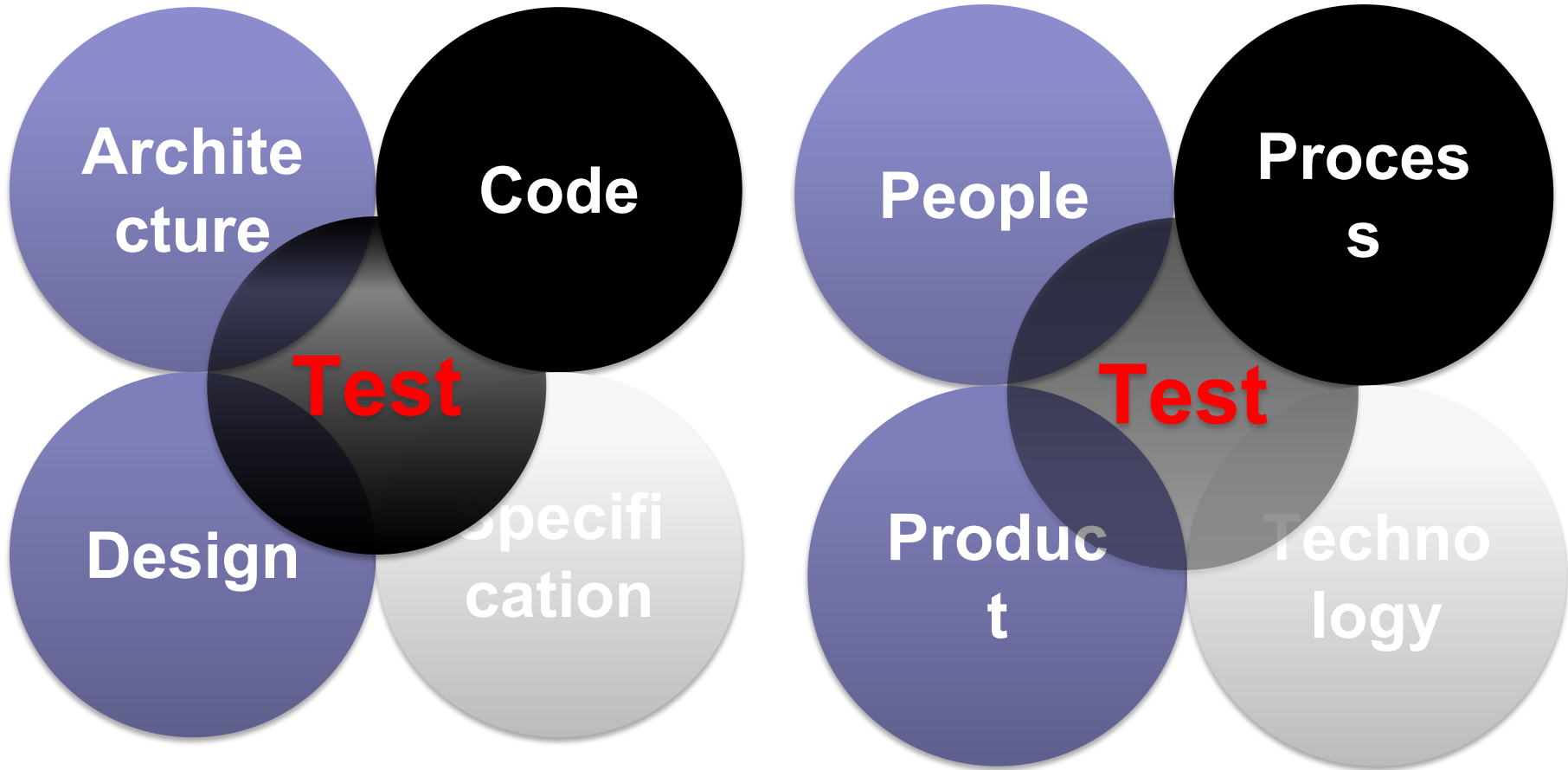
education

Analysis

services

environment

Building Blocks





Software Engineering

IEEE Definition

Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of Software; that is the application of engineering to software (2) The study of approaches as in (1).

Software Engineering is the establishment and use of a sound engineering principle in order to obtain economically software that is reliable and works efficiently on real machine. **[Fritz Bauer]**

- **Where did Software Engineering come from?**
- **Rather why was there a thought of Software Engineering?**

Goals of Software Engineering



1. To improve quality of software
2. To improve the productivity of developers and software teams

And many more...



Software Quality Attributes

Types of attributes:

- **Static:** structured, maintainable, testable code as well as the availability of correct and complete documentation.
- **Dynamic:** software reliability, correctness, completeness, consistency, usability and performance

Attributes:

- Completeness
- Consistency
- Usability
- Performance



What if testing does not detect any errors

Either

- the software is high quality

Or

- the testing process is low quality

Metrics are required on our testing process if we are to tell which is the right answer.



Test Techniques

Based on Engineers experience and intuition

- Exploratory
- Ad-hoc

Specification Based Techniques

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Tables
- ...

Code Based Techniques

- Control Flow
- Data Flow
- ...



Test Techniques



Techniques based on nature of application

- Object Oriented
- Component-based
- Web-based
- GUI
- Protocol Conformance
- Real Time Systems

Usage based testing

- Operational Profile
- Reliability Engineered Testing

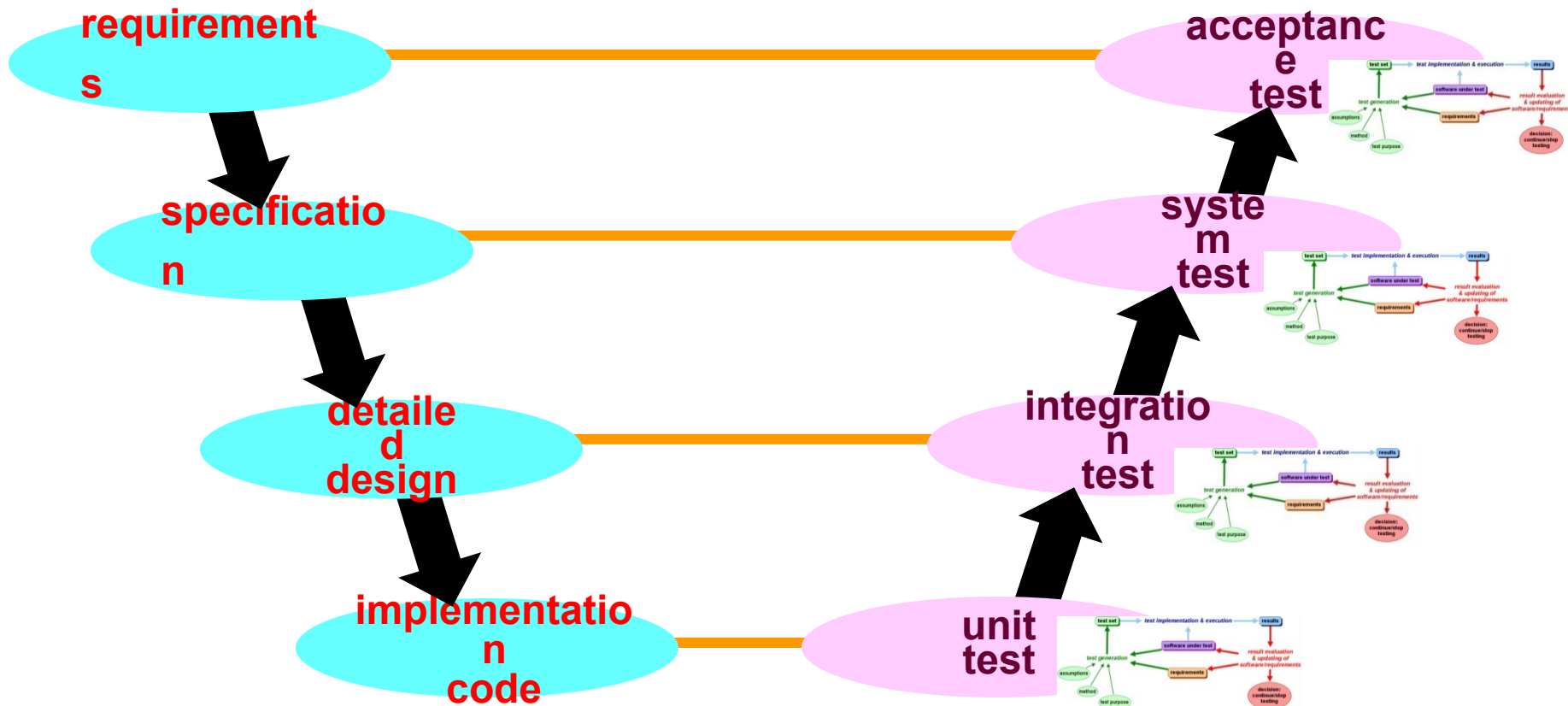
Goals of SW Testing



- To show that the software system satisfies the requirements and performs as expected. The requirements may be explicit or implicit.
 - Explicit: User Interface, Specified Output
 - Implicit: Error handling, performance, reliability, security
- To have “confidence” in the software system. To assure and demonstrate that the Software works.
- To find defects
- To prevent defects
- Ensure software quality

Test process in software development

V-model:



What is...

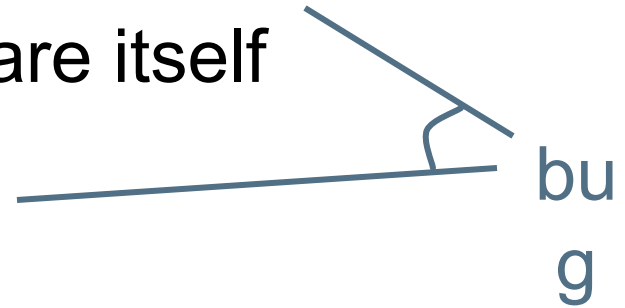
error: something wrong in software itself

fault: manifestation of an error

(observable in software behavior)

failure: something wrong in software behavior

(deviates from requirements)



requirements:

for input i ,
give output $2 \cdot (i^3)$

(so 6 yields 432)

software:

error
 $i = \text{input}(\text{STDIN});$
 $i = \text{double}(i);$
 $i = \text{power}(i, 3);$
 $\text{output}(\text{STDOUT}, i);$

output (verbose):

input: 6
doubling input..
computing power..
output: 1728
fault

fault + failure

What is...

testing:

by experiment,

- find errors in software (Myers, 1979)
- establish quality of software (Hetzel, 1988)

a successful test:

- finds at least one error
- gives quality judgment with maximum confidence with minimum effort

Possible Goals of Testing

- Find faults
 - Glenford Myers, *The Art of Software Testing*
- Provide confidence
 - of reliability
 - of (probable) correctness
 - of detection (therefore absence) of particular faults



Testing and Debugging

- **Testing** is the process of determining if a program has any errors.
- When testing reveals an error, the process used to determine the cause of this error and to remove it, is known as **debugging**.



Test Plan

- A test cycle is often guided by a **test plan**.
- A **test case** consists of test data to be input to the program and the expected output.
- The **test data** is a set of values, one for each input variable.
- A **test set** is a collection of zero or more test cases.

Correctness versus reliability



- **Correctness** of a program is desirable, it is almost never the objective of testing.
- To establish correctness via testing: test a program with all possible inputs— this is impossible to accomplish.
- Completeness of testing does not necessarily demonstrate that a program is error free.
- **Reliability:** Probability of failure free operation of software over a given time interval and under given conditions. It is the probability of failure free operation of software in its intended environment.



Testing and Verification

- **Program verification** aims at proving the correctness of programs by showing that it contains no errors. This is very different from testing that aims at uncovering errors in a program.
- Program verification and testing are best considered as complementary techniques. In practice, program verification is often avoided, and the focus is on testing

Testing Principles

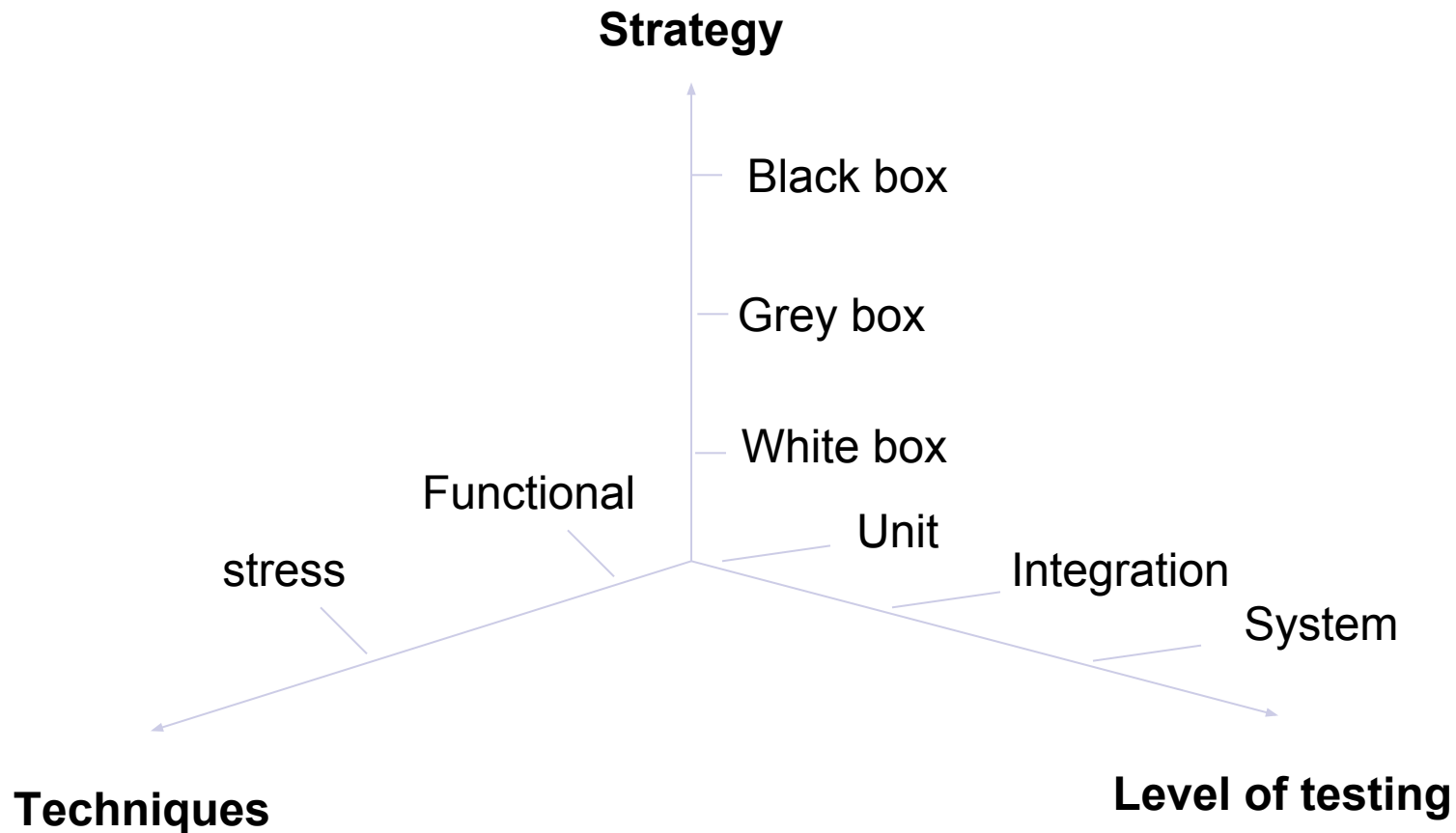
- A Test case must include the definition of expected results
- A programmer should not test her/his own code
- A programming organization should not test its own programs
- Results of each test should be thoroughly inspected
- Test cases must be written for invalid inputs also
- Check that programs do not show unexpected behavior
- Do not plan testing assuming that there are no errors
- The probability of error in a piece of code is proportional to the errors found so far in this part of the code

Standard Testing Questions

- Did this test execution succeed or fail?
 - Oracles
- How shall we select test cases?
 - Selection; generation
- How do we know when we've tested enough?
 - Adequacy
- What do we know when we're done?
 - Assessment

Thumb rule for Testing

- Testing is based on 3 major factors

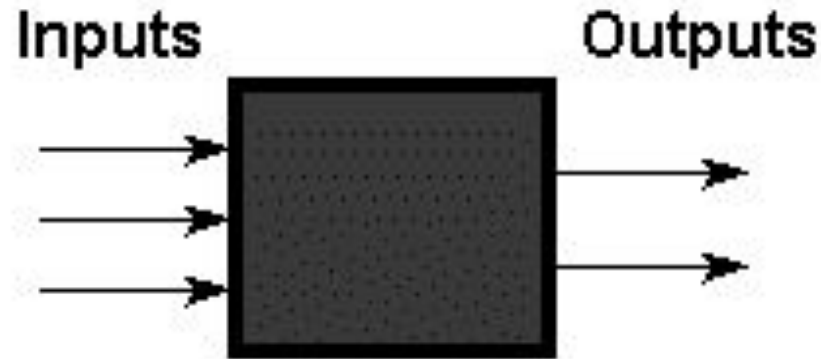


Test generation strategies

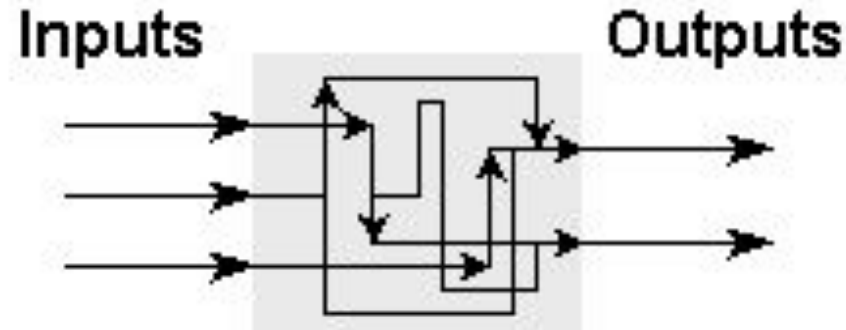
- **Model based:** require that a subset of the requirements be modeled using a formal notation (usually graphical).
 - **Models:** Finite State Machines, Timed automata, Petri net, etc.
- **Specification based:** require that a subset of the requirements be modeled using a formal mathematical notation. Examples: B, Z, and Larch.
- **Code based:** generate tests directly from the code.

Strategies of testing

Black box



White box





Black and White Box Testing Methods

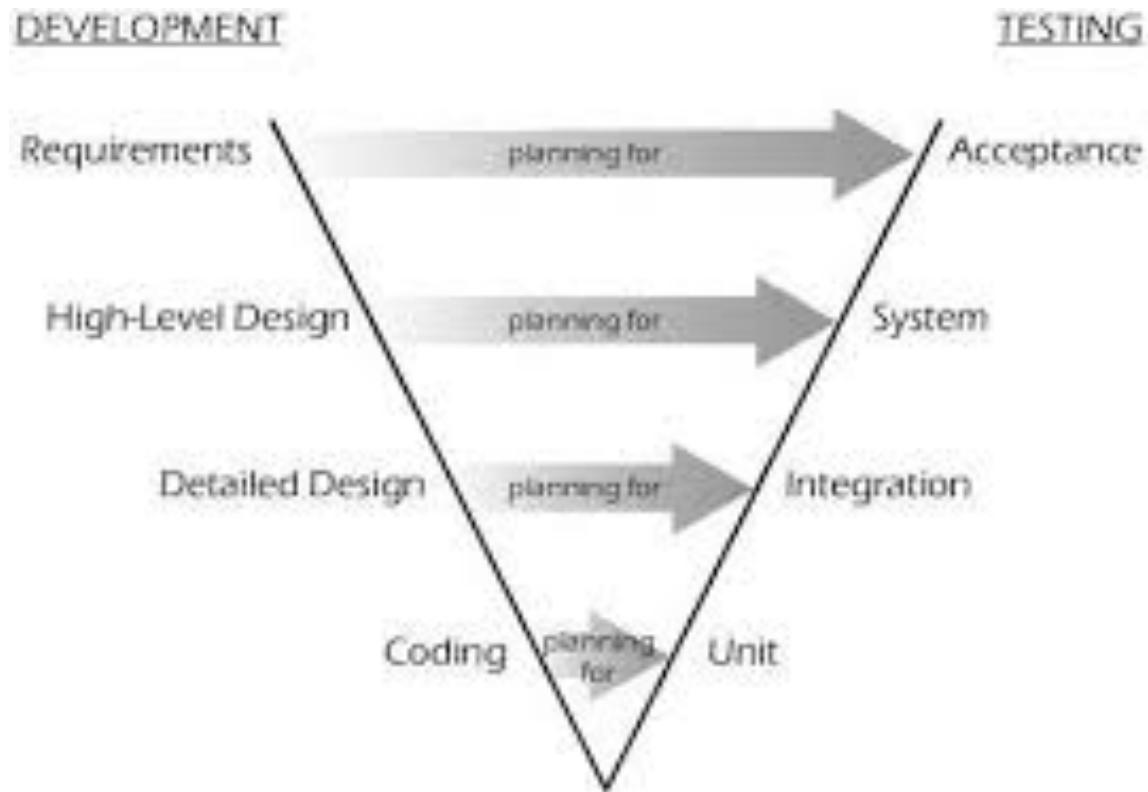
Black box

- Equivalence partitioning
- Boundary-value analysis
- Cause-effect graphing
- Error guessing
- State space exploration

White box

- Coverage/Adequacy
- Data flow analysis
- Cleanness
- Correctness
- Mutation
- Slicing

Levels of abstraction and testing in waterfall model



Ten dominating techniques

- **Function testing**

-
-
-
-
-
-
-
-
-

Test each feature or function on its own.

Scan through the product, covering every feature or function with at least enough testing to determine what it does and whether it is working.

Ten dominating techniques

-
- **Specification-based testing**
-
-
-
-
-
-
-
-

Check every claim made in the reference document (such as, a contract specification).

Test to the extent that you have proved the claim true or false.

Ten dominating techniques

-
-
- **Domain testing**
-
-
-
-
-
-

Focus on variables, such as inputs, outputs, configuration, or internal (e.g. file-handling) variables.

For every variable or combination of variables, consider the space of possible values. Simplify it by partitioning into subsets. Pick a few representatives of each subset.

Ten dominating techniques

-
-
-
- Risk-based testing
-
-
-
-
-
-

A program is a collection of opportunities for things to go wrong.

For each way that you can imagine the program failing, design tests to determine whether the program actually will fail in that way.

Ten dominating techniques

-
-
-
-
- **Scenario testing**
-
-
-
-
-

Tests are complex stories that capture how the program will be used in real-life situations.

These are combination tests, whose combinations are credible reflections of real use.

Ten dominating techniques

-
-
-
-
-
- Regression testing
-
-
-
-

Repeat the same test after some change to the program.

You can use any test as a regression test, but if you do a lot of regression testing, you will (or should) learn to design cases for efficient reuse.

Ten dominating techniques

-
-
-
-
-
-
-
-
-
-

Stress testing

Many definitions of stress testing.

When I say stress testing, I mean tests intended to overwhelm the product, to subject it to so much input, so little memory, such odd combinations that I expect it to fail and am exploring its behavior as (and after) it fails.

Ten dominating techniques

-
-
-
-
-
-
-
-
-
-

User testing

Give the program to “a user,” see what he does with it and how it responds.

User tests can be tightly structured or very loosely defined. The essence is room for action and response by “users”.

Ten dominating techniques

-
-
-
-
-
-
-
-
- **State-model based testing**
-

Model the program as a state machine that runs from state to state in response to events (such as new inputs).

In each state, does it respond correctly to each event?

Ten dominating techniques

-
-
-
-
-
-
-
-
-

Program the computer to design, implement, execute and interpret a large series of tests.

You set the wheel in motion, supply the oracle(s) and evaluate the pattern of results.


- **High volume automated testing**

Coming to Grips With Reality

- Software will be shipped with bugs
 - Known and unknown issues
 - Due to time constraints, lack of test coverage
- Users will do things the designers never imagined or intended
 - Either accidentally or cleverly
 - With benign and malicious intent
- Plan to fix bugs after a release
 - Software is never “done”
 - Must plan ahead for releasing updates

Generalized Pseudocode

- Pseudocode provides a *language-neutral* way to express program source code.
- Language element – Generalised Pseudocode examples:
 - Comment - '<text>
 - Data Structure declaration - Type <type name>
 <list of field descriptions>
 End <type name>
 - Assignment statement - <variable> = <expression>
 - Input - Input (<variable list>)
 - Output - Output (<variable list>)

- 
- The triangle problem
 - The NextDate function
 - The commission problem
 - The SATM (Simple Automatic Teller Machine) problem
 - The currency converter problem
 - Saturn wind shield wiper problem