Software testing

Basic concepts

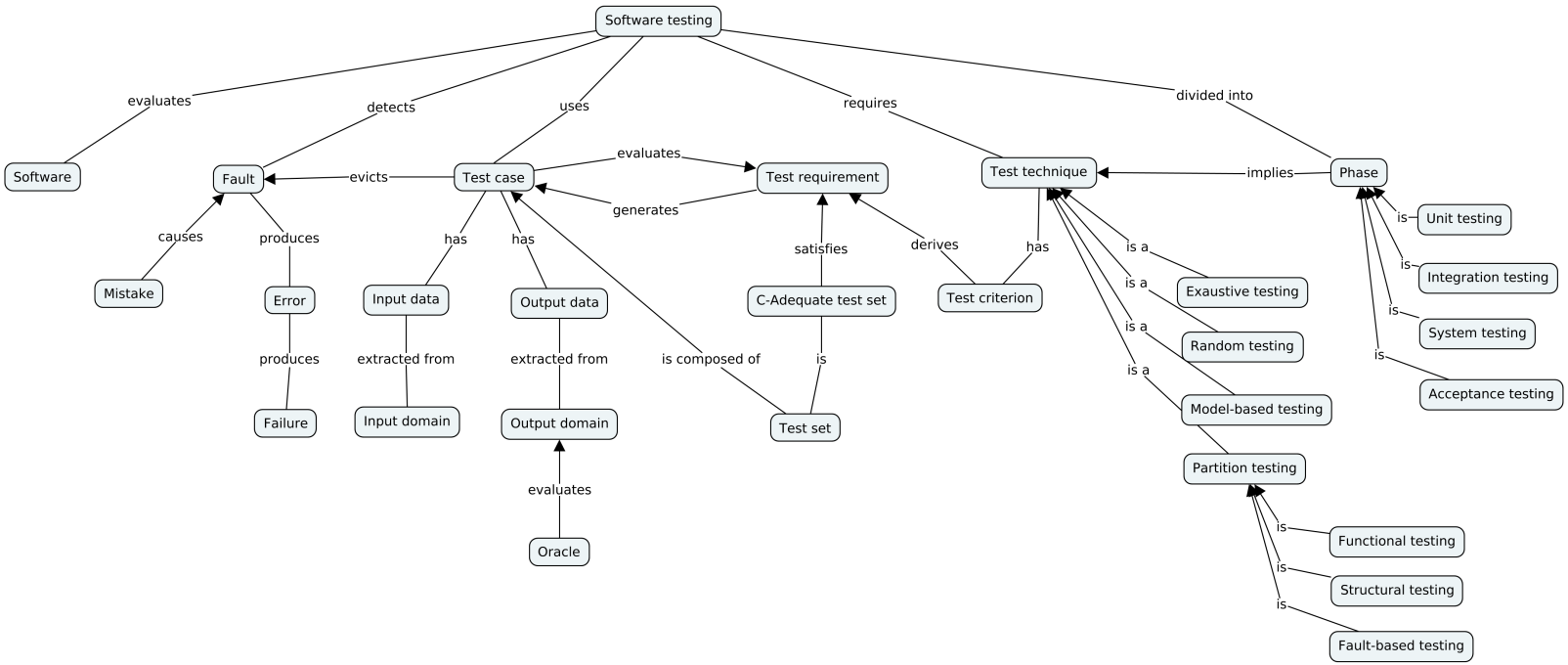
Marco Aurélio Graciotto Silva[1], Ellen Francine Barbosa[2], José Carlos Maldonado[2]

[1]**Department of Computing**
Federal University of Technology – Paraná (UTFPR)
Campo Mourão, PR, Brazil

[2]**Institute of Mathematical Sciences and Computing**
University of São Paulo (USP)
São Carlos, SP, Brazil

2017

Software Testing



Courses-SoftwareTesting-BasicConceptsOfSoftwareTesting

Defect

Defective products

- No product is defect-free.
  - And that applies to any kind of product, be it software or hardware.

Example: Product failures

Defective software

- No software module is defect-free when implemented and only 40% of software modules are defect-free when released [**?**].

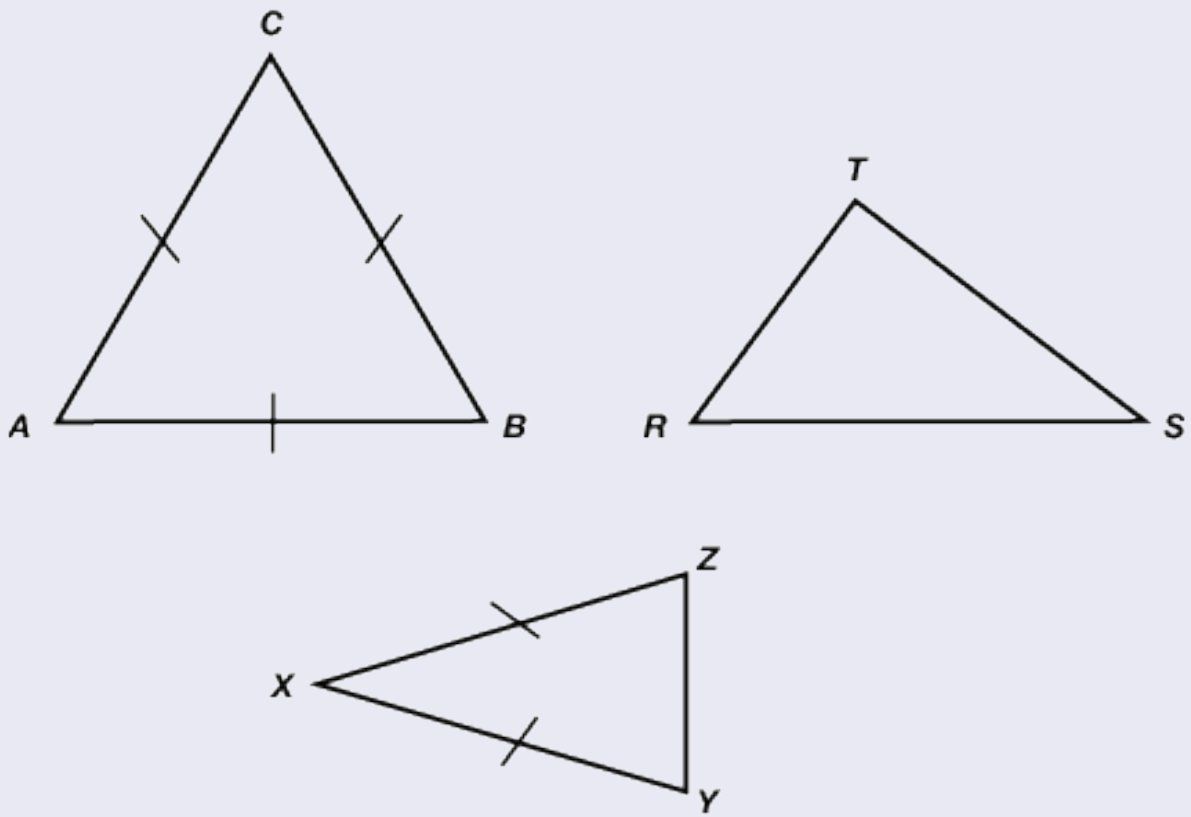
Example: Software failures

Defect

Every software has defects

- Even small, simple, trivial products are error-prone.

Example: Triangle



Defect

Defect avoidance and detection

- Nevertheless, products failures could and must be avoided and corrected, as they negatively impacts software quality.

Why we should care about it?

- Rising demand for higher software quality.
- Software defect reduction improves software quality.

Software quality and software testing

- The main goal of software testing is to find defects.
- Systematic software testing, carried out using proper techniques, criteria, and tools, improves software reliability (and quality).

Motivation

- Software contains fault.
- Testing the software can reveal several faults.

Software testing as a discipline

- *Ad hoc* testing is insufficient and ineffective at fault detection.
  - It is hard to think of enough test cases even for simple software (such as Triangle).
- Software testing must be taken as a discipline.
  - Test cases must be designed to assess the quality of software artifacts (mainly the requirements specification and the source code) regarding given criteria.
  - Techniques must be developed to detect as much faults as possible in the software.

Definition

Software testing is the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component [**?**].

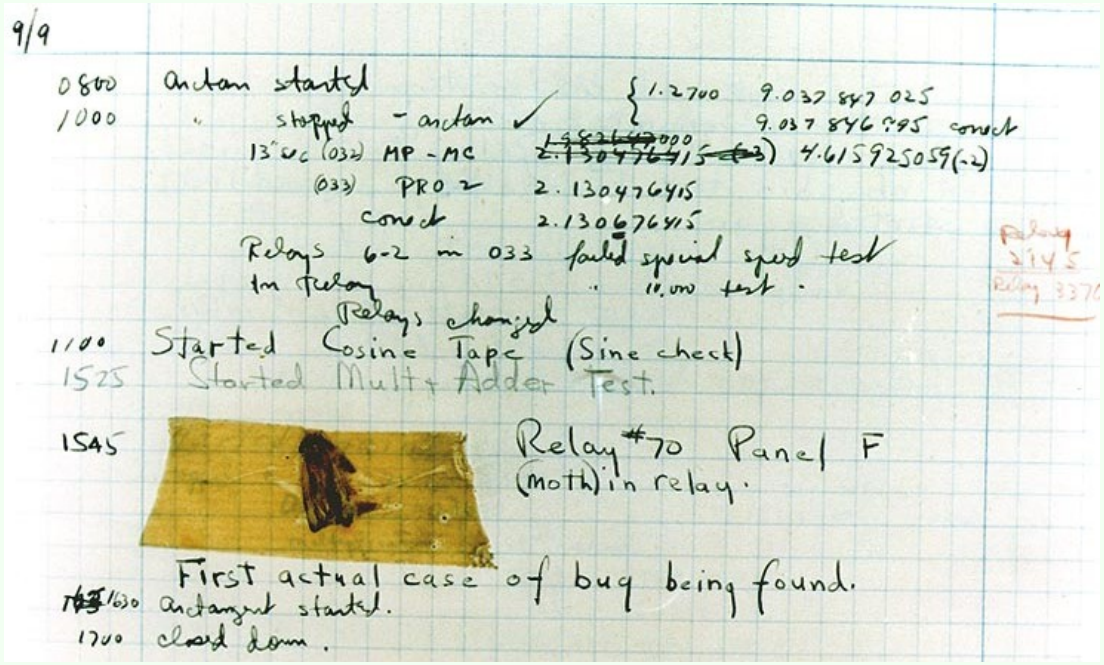
Software testing is. . .

- a verification and validation activity,
- important for maintenance, reliability assessment,software process improvement, debugging,
- a dynamic activity,
  - It requires the execution of the program (static analysis is not enough),
- expensive [**?**]

Goal

- The goal of software testing is to detect faults in the product under testing.

First error ever detected (by Grace Hopper). It is an actual bug (that is why we call errors of bugs till today).



Contradiction

- Thus, software testing aims at virtually destroying the software?
  - After all, the programmer spent hours implementing it and test activities will find errors in the work just done by him.

Rationale

- Humans are highly goal-oriented (and proper goals has an important psychological effect [**?**, p. 6]).
- If the goal is to demonstrate that a program has no error, then the tester will subconsciously be steered toward this goal.
  - There will be a tendency to select test data that have a low probability of causing the program to fail.
- If the goal is to demonstrate that a program has error, the test data will have a higher probability of finding error.

Rationale

The process of demonstrating that errors are not present is impossible to achieve for virtually all programs (even trivial ones).

Undecidable problems

- The assessment of the correctness of a program is an undecidable problem.
  - Undecidable = non-computable.
- Errors can be masked by other errors due to coincident correction.
  - And assessment of coincident correct is also an undecidable problem.
- Several other undecidable problems plague software testing: software equivalence, executability.

How are faults detected?

- Faults are detected by running the software against a set of test cases.

Test case execution

1. Define the input data to be fed to the software under testing.
2. Run the software.
3. Check if the result the software produced was the expected one (for the given input data).
   - If the result is correct, keep defining different input data.
   - If the result is incorrect, a fault was found! Success!

Exhaustive testing

If the software is executed against all possible input data, any fault will be found!

Feasibility and computability

Unfortunately, it is often not possible to run exhaustive testing due to some computational problems:

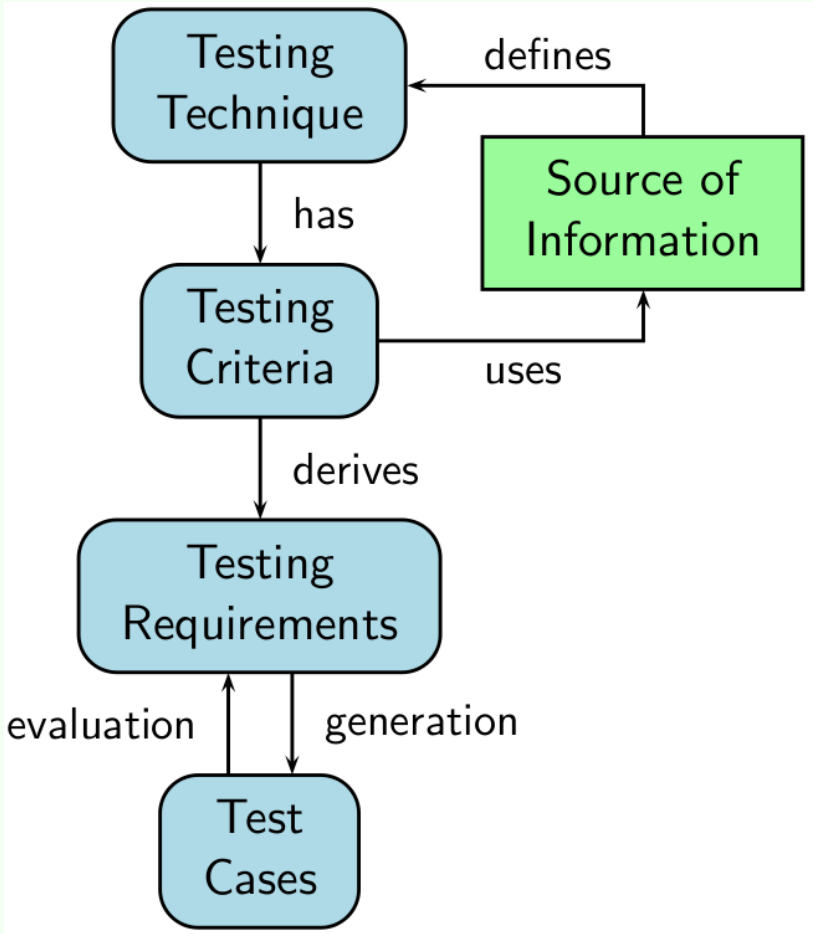
- the input domain may be so big that it is impossible to run the software against it in a reasonable time,
- computability (undecidable problems).

Domain partitioning

Nonetheless, it is possible to partition the input domain and, instead of using all elements within each partition, just a few ones are selected (and accepted as a good representation of every element in the set).

Techniques and criteria

- Test criterion defines rules regarding how a given input domain is partitioned.
- Such rules, when applied to the software under testing, generates test requirements.
  - Test requirement is a combination of elements of the program under testing that must be satisfied by the execution of a test case.
- Test technique defines the rationale and source of information that guides the development of test criteria.



What is a fault?

A fault is an incorrect data definition, step, or process in a software.

How a fault is created?

- A fault is inserted by a mistake.

Example: Incorrect statement

What is a mistake?

A mistake is a human action that produces an incorrect result.

How a mistake takes place?

- Lack of attention when implementing the software?
- Omitted information in the software requirement specification?
- Compiler fault (which by itself was caused by a mistake)?

Example: Incorrect statement

What is an error?

An error is the difference between a computed, observed or measured value or condition and the true, theoretically correct or specified value or condition.

How an error occurs?

- An error is produced by the execution of a fault.
  - Thus, if a fault is never executed, it never produces an error.

What is a failure?

A failure is the inability of a component or a system to fulfill its required functions within specified performance requirements.

How a failure occurs?

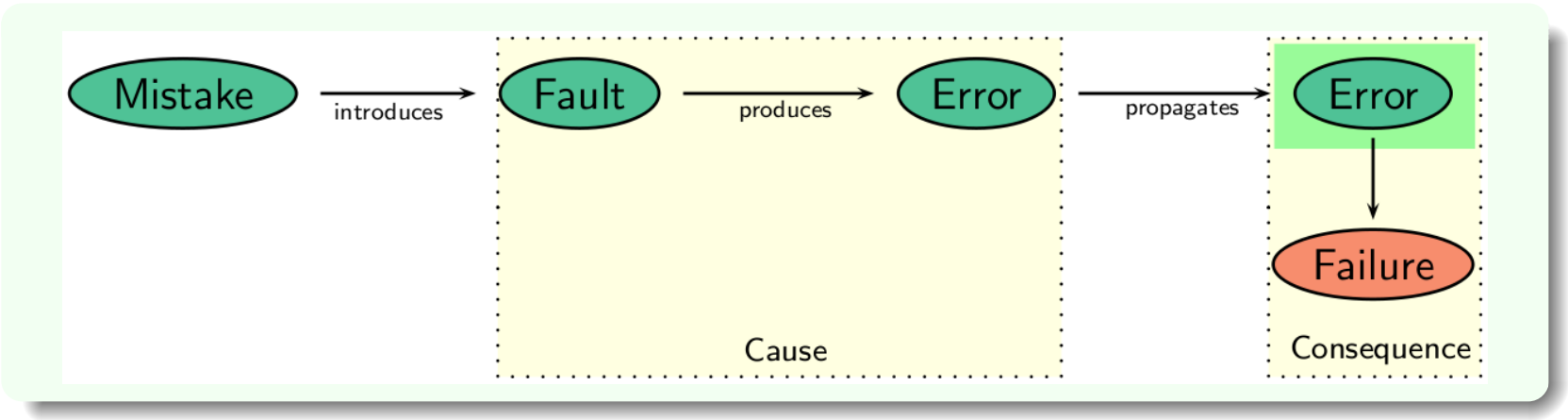
- A fault and its associated errors may cause one or more failures.

Example: Incorrect statement

Summary

1. A programmer makes a **mistake**.
   - One single $<$ is replaced by a $>$ in a single statement.
2. Due to the mistake, the statement is **faulty** (it does not implement the behavior described in the software requirements specification).
3. The software is compiled and run. The given statement is run. As it belongs to a function in charge of creating the checksum of the data being processed, the output is produces is incorrect (**error**).
4. The user compares the checksum produces by the software with the expected one. No matter what, the checksum always **fails**.

Defect taxonomy



Example: Physician analogy for defect taxonomy

Example: Defect taxonomy example (numZero)

Test case

Simplified definition

A test case is a pair consisting of test data (a set of values, one for each input variable) to be input to the program and the expected output.

A better definition

A test case is usually defined as a tuple $(d, S(d))$, where:

- $d \in D$ (and D is the input domain), and
- $S(d)$ represents the expected output for the input d according to specification S.

Example: Test cases for a sort method
Example: Test cases for the numZero method

Successful test cases

- At first, a well-constructed and executed test case is successful when it finds errors [**?**, p. 7]
- It is also successful when it eventually establishes that there are no more errors to be found (as when applying a test criteria and satisfying all the test requirements).

Unsuccessful test cases

- A unsuccessful test case is one that causes a program to product the correct result without finding any error.

Analogy for test cases success and failure

Execution order

- There are two styles of test case design regarding order of test execution:
  - cascading test cases, and
  - independent test cases.

Definition

Cascading test cases are test cases that build on each other.

Advantages and disadvantages

- The advantage of cascading test cases is that each test case is typically small and simple.
- The disadvantage of cascading test cases is that if one test fails, the subsequent tests may be invalid.

Example: Cascading test cases

Definition

Independent test cases are entirely self contained.

- Independent test cases neither build on each other nor require that other tests have been successfully executed.

Advantages and disadvantages

- The advantage of independent test cases is that any number of tests can be executed in any order.
- The disadvantage of independent test cases is that each test tends to be larger and more complex and thus more difficult to design, create, and maintain.

Oracle

Is it correct?

- Given a set of input conditions and the observable results of the computation, who decides whether the results are correct?
- Somebody or something must check whether the software, for a given test case, has operated correctly.

Oracle

Definition

An oracle is any software, process, or data that provides the test designer with the expected result of a test case.

- An oracle decides whether output values are correct against what is specified.
  - An oracle is required to determine whether a fault was revealed.
- Some examples of oracle: human guess (kiddie oracle), regression test suite, validated data, purchased test suite, existing software.

Definition

A kiddie oracle is obtained from running the software and seeing the output. If it looks about right, it must be right.

Why should I use a kiddie oracle?

- Actually, you should not use it, as it is error prone.
- Nonetheless, it is better than nothing.
  - And, if the expected behavior of the application is not documented, it is up to the user to detect the correct result anyway.

Example: Human guess (kiddie) oracle

Definition

A regression test suite oracle is obtained from running the test case and comparing the output to the results of the same test cases run against a previous version of the software.

Why should I use a regression suite oracle?

- Regression test ensures that the modified system functions as per its specification.
- It ensures that old errors will not appear again (or that, at least, they will be early detected).

Example: Regression test suite oracle for Mozilla Firefox

Definition

A purchased test suite oracle consists in running the software against a standardized test suite that has been previously created and validated.

Why should I use a purchased oracle?

- Usually it is required that a software passes a test suite oracle in order assess its compliance to a particular technology or standard.
- Sometimes its simply easier to buy a test suite instead of developing your own.

Example: Java TCK

What I should test first?

- What should I target when testing a software?
- The definition of a single target for each test activity is key for systematic software testing.
  - Focus effort and restrict available techniques (thus lowering costs and, probably, increasing efficacy).
- A common sense approach would be to begin testing the smallest module possible, and then proceed to test the interactions between these modules and, finally, the system as a whole.

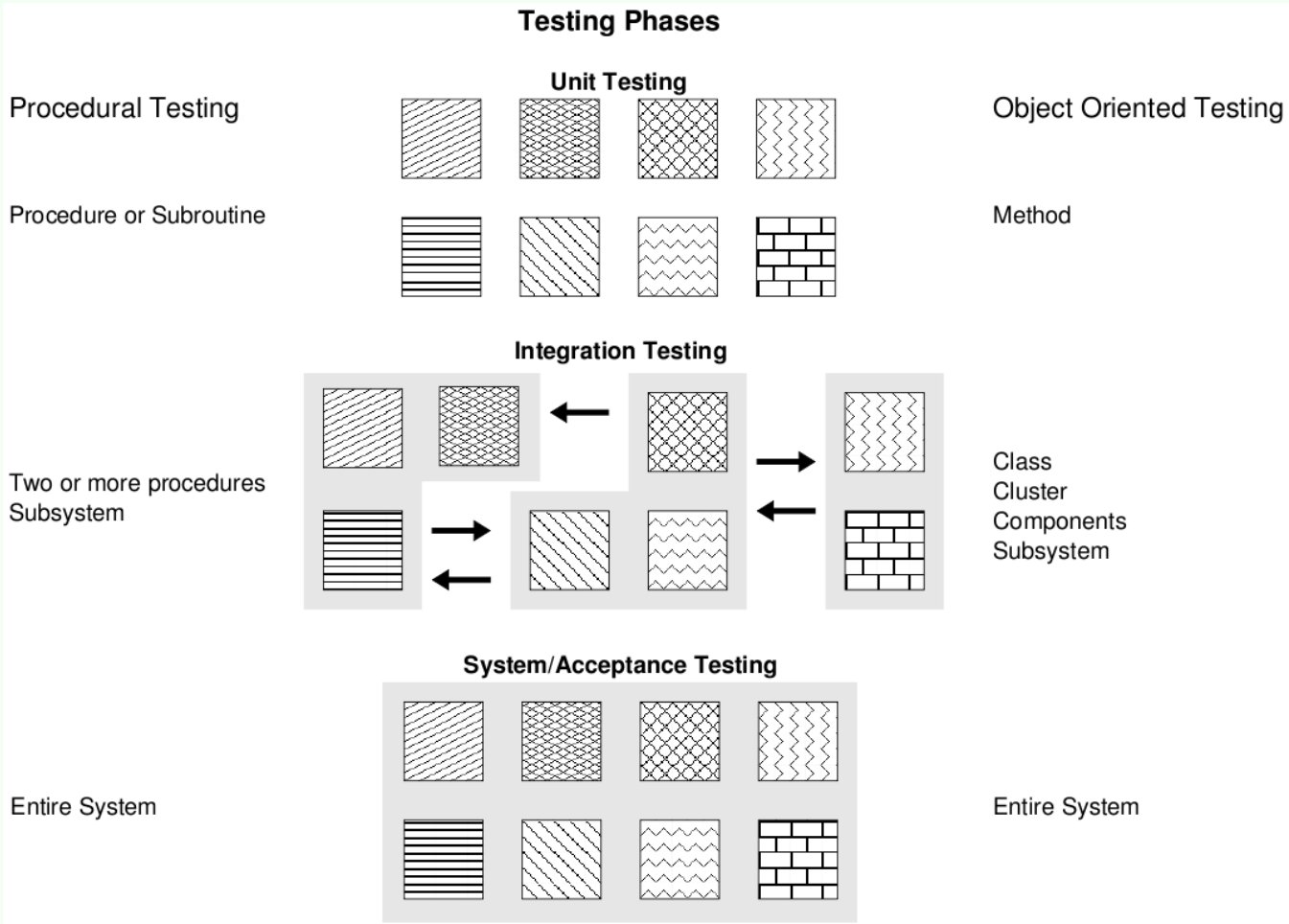
Definition

Test phase is a categorization of test activities which is directly related to the software life cycle the activities take place in.

Definitions

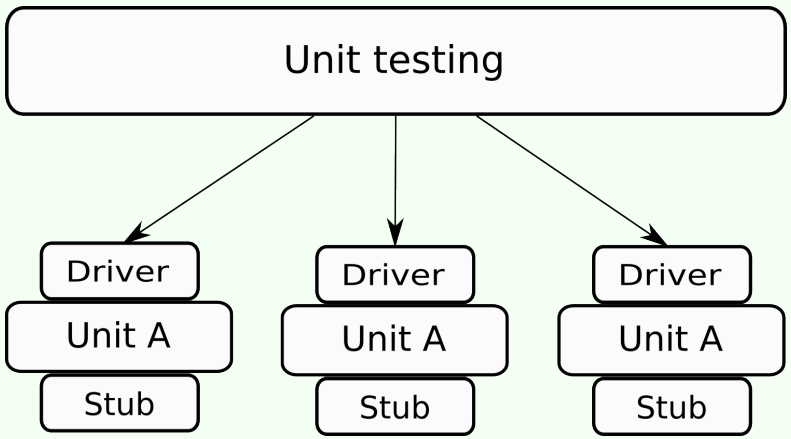
- Test phases allow the tester to concentrate on various aspects of the software and use different test criteria at each one.
- Test activities are organized in test phases such that testing starts with the smallest executable unit until reaching the software as a whole:
  - Unit testing.
  - Integration testing.
  - System testing.
  - Acceptance testing.

Test phases



Definition

Unit testing verifies the functioning in isolation of software pieces which are separately testable.



Example: Pentium FDIV bug

What is an unit?

- In procedural testing, the unit is the procedure or subroutine.
- In object oriented testing, the unit is the method or the class.
  - For the time being, we will consider that the unit is the method!

How to test an unit?

- Although the definition calls for units that are separately testable, this is not always possible.
  - Mosts units requires data from other units.
- Coupling between units is a threat for unit testing. Thus, it is desirable to replace some units by others more simple and predictable (just for software testing sake).

Definition

A stub is a unit that replaces another unit used by the unit under test.

Why should we use a stub?

- Stubs simulate the behavior of another unit not yet implemented but called by the unit under test.
- Usually, a stub simulates the expected behavior of the used unit with minimum computation effort or data manipulation.

Example: Stub

How to test a unit?

- Even though stubs can be used to replace complex units for a given unit under testing, something must setup and wire the stubs into the unit.

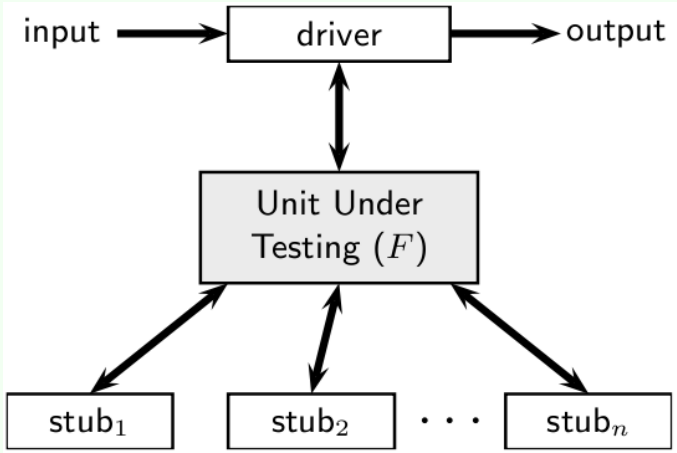
Definition

Driver is the software responsible for coordinating the testing of a unit.

What a driver does?

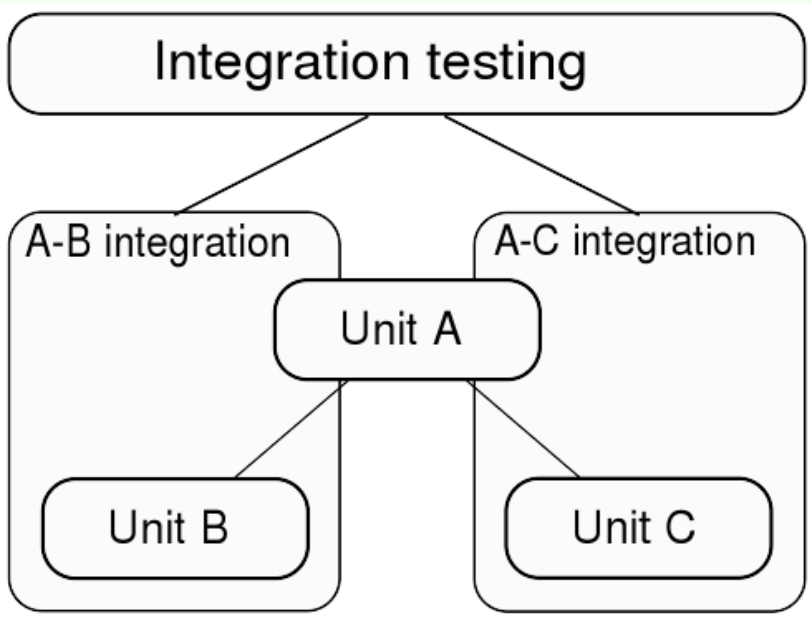
- Drivers are used to test a unit which requires input data provided by another unit:
  1. it gathers the data provided by the tester,
  2. it passes them to the unit under test in the form of arguments,
  3. it collects the results produced by the unit, and
  4. it shows them to the tester.

Test phases
Unit testing - Driver and stubs



Definition

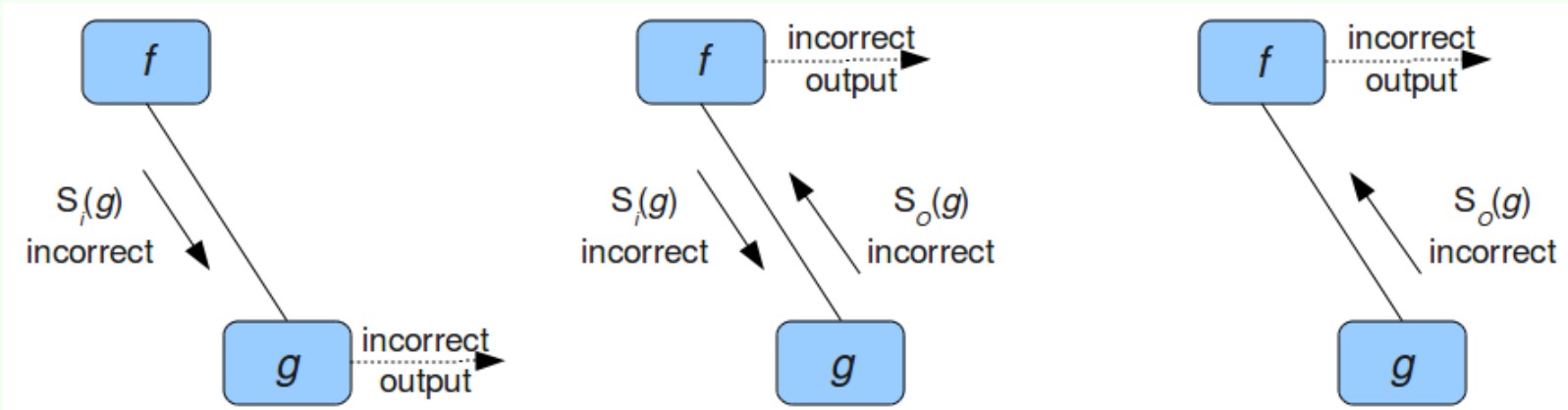
Integration testing verifies whether the units tested individually communicate accordingly when integrated.



Example: Mars climate orbiter

Why integration testing is important?

- Integration testing must be performed because:
  - Data can be lost at the unit's interface.
  - Global variables can suffer undesirable interferences.



Definition

System testing ensures that the software and the other elements that are part of the system (hardware and database, for instance) are adequately combined and behave as expected.

Types of system testing

- Typically system testing includes many types of testing:
  - functionality,
  - usability,
  - security,
  - localization,
  - reliability,
  - availability,
  - ...

Example: Ghost train

Definition

Acceptance testing refers to the tester, in general performed by the user himself, who verifies whether the product satisfies his expectation.

Alpha/Beta testing

- Informally, it can be defined alpha and beta testing.
  - Alpha testing: software is installed and used internally (in the company that developed).
  - Beta testing: software is installed and tested by external users.

Why is acceptance testing important?

- Acceptance testing, when completed successfully, will result in the customer accepting the software.

Test techniques and criteria

- Test techniques provides a grounded theory on how we should test our software:
  - What are the sources of test requirements?
  - What features of the source should be exploited?
- However, in order to successfully test a software, it must be throughly specified how the techniques should be applied.
  - And by successfully we mean: to find errors.

How to increase efficacy

- Test requirements should be established so they explore not only what the software should do, but also what it should **not** do.
- A measure should also be provided, so that a stop criteria can be defined for the test activity.

Definition

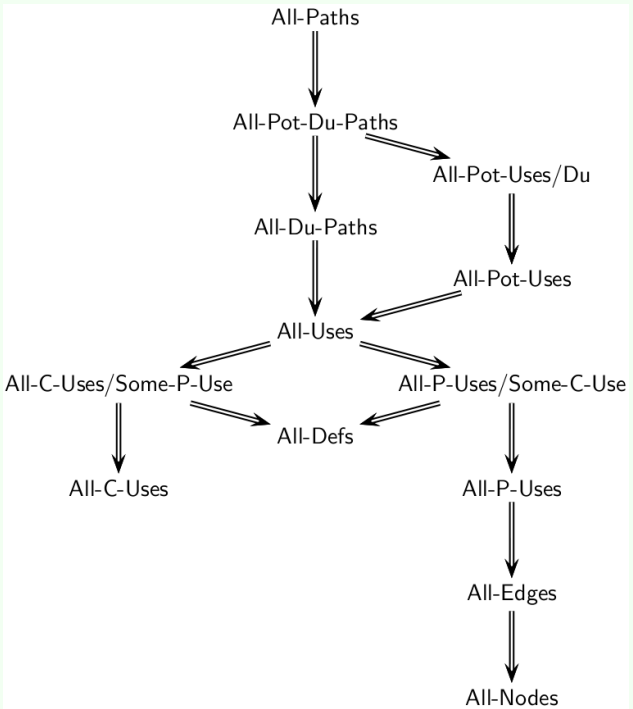
A test criterion systematizes the way test requirements are generated from the source of information (specification, source code, historical fault database, etc.).

- A test criterion provides a systematic way to select test cases.
  - A test criterion divides the input domain.
- When no faults are found, test criterion provides an indication of how test cases should be selected in order to establish a high level of confidence of product correction.

Example: Test criterion example

Test criterion attributes

- A test criterion can be compared based on cost, efficacy and strength.



Test criterion

Test coverage

- A test criteria can be used either to evaluate the program under testing or to evaluate the thoroughness of the test set.

Test selection criterion

The test criterion is used to select test cases to evaluate the program under testing.

Test evaluation criterion

The test criterion is used to evaluate the test sets.

Test requirement

Definition

A test requirement is a specific element of a software artifact that a test case must satisfy or cover.

How a test requirement is created?

- Test requirements are derived from the program under testing using a specific test criterion.

What are test requirements for?

- A test requirement can:
  - evaluate a test set, and
  - generate a test set.

Definition

A test set is a set of test cases.

Test sets and test requirements

- A test set can be improved by adding test cases that exercise uncovered requirements.
  - The best test set is the smallest one that indicates the largest set of faults.

C-adequate test sets

- When a test set T satisfies all the test requirements derived from a program using a given criterion C, T is said $C - adequate$.

Definition

Test techniques are types of testing defined according to the source of information used to carried out the testing activity.

Test techniques and test criteria

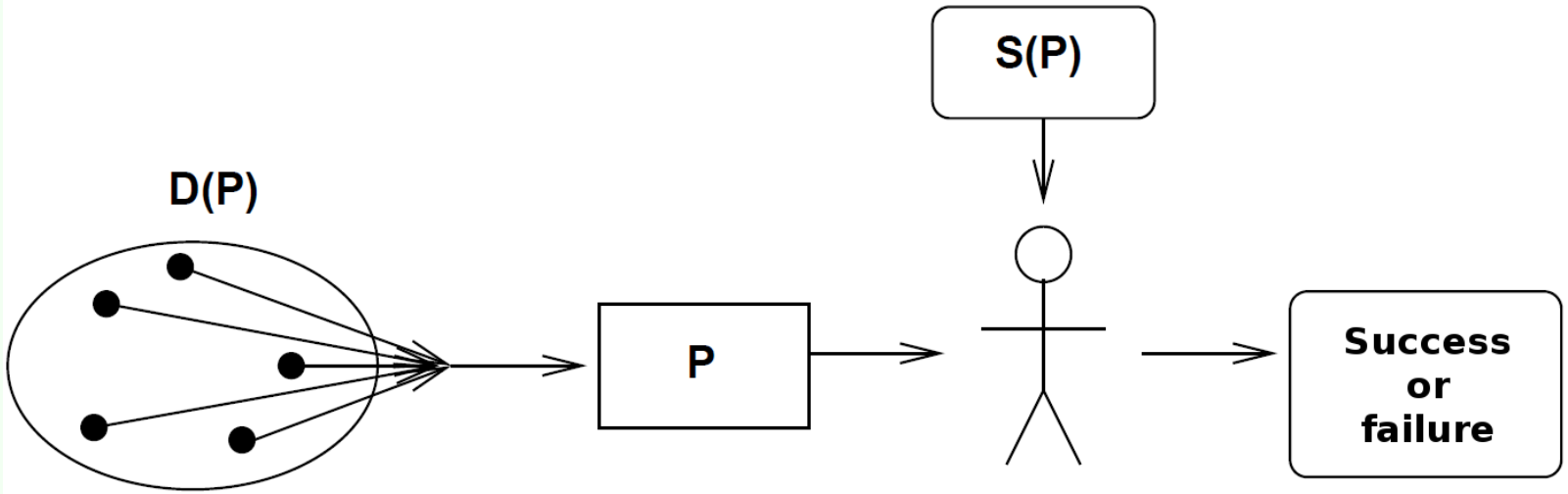
- Each test technique has a set of associated test criteria.

Software test techniques

- Exhaustive testing
- Random testing
- Partition testing
  - Fault-based testing
  - Functional testing
  - Structural testing

Definition

An exhaustive test executes the software with all possible value from its input domains.



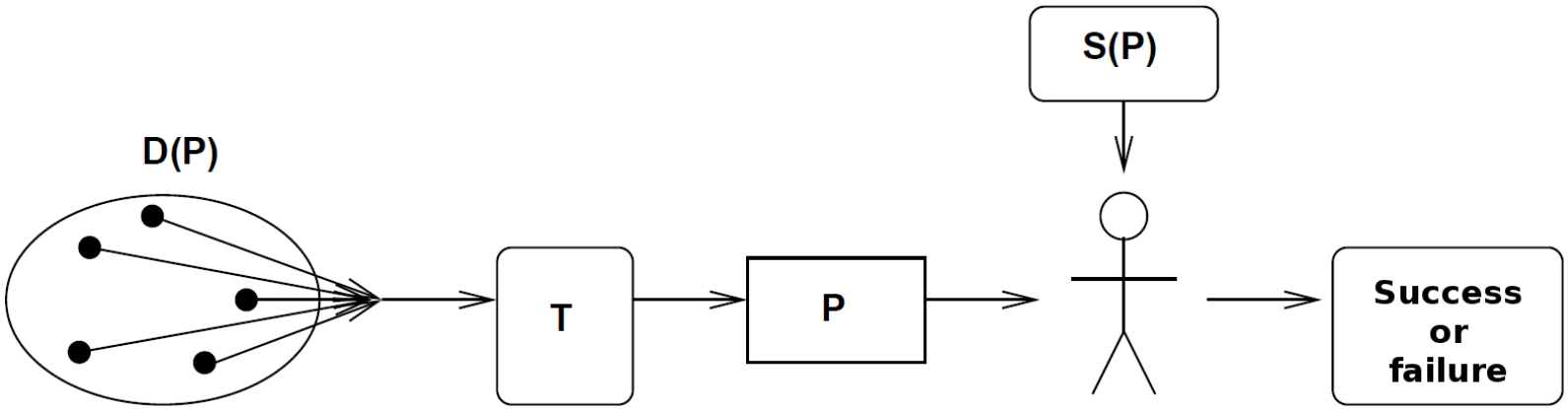
Example: Exhaustive testing of the blech function

Exhaustive testing limitations

- Can be prohibitive due to time and cost constraints for finite but large input domain.
- Impossible if the input domain is infinite.
- Infeasible in general.

Definition

Random testing uses a systematic method to generate test cases: it requires modeling the input space and then sampling data from the input space randomly.



Reliability

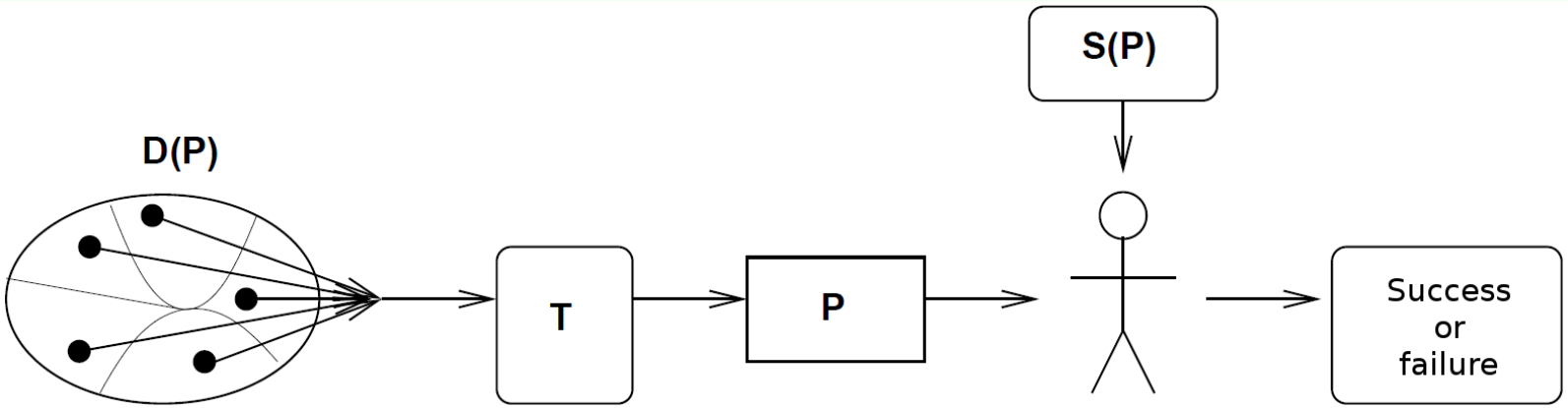
- Using random testing, statistical measures of reliability can be achieved according to an operational profile.
- For every (input data of a) test case, it is assigned a probability distribution according to their occurrence in actual operation.

Effectiveness

- It depends on the correctly definition of the operational profile.
- If the probability of occurrence of each input data is the same, random testing is regarded as the least effective technique for software testing [**?**, p. 43].

Definition

Partition testing is meant as any testing scheme which forces execution of at least one test case from each subset of a partition of the input domain.



Definition

Functional testing is a technique based solely on the requirements and specifications.

- Functional testing is also known as black box testing.
- Functional testing obtains test requirements from the software specification.
  - Functional testing requires no knowledge of the internal paths, structure, or implementation of the software under test.

Example

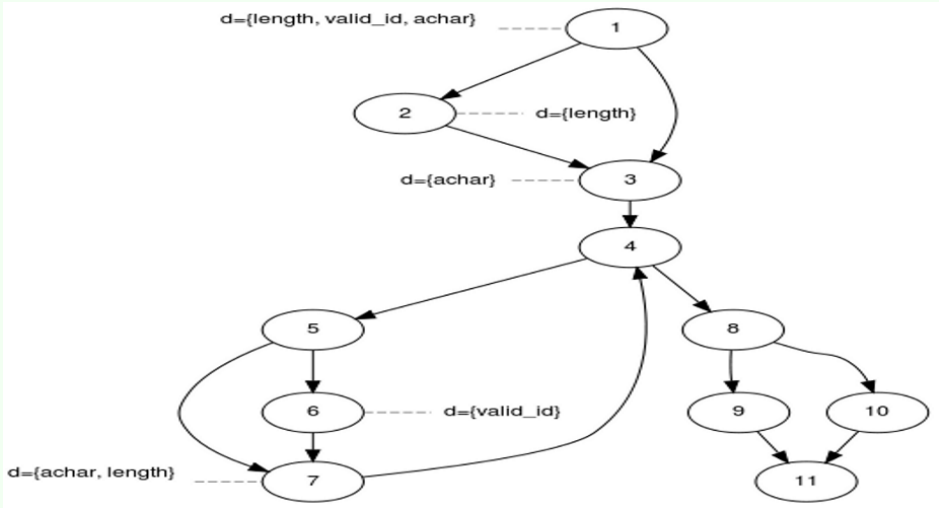
Functional test criteria

- Equivalence partition.
- Boundary-value analysis.
- Cause-effect graph.
- ...

Definition

Structural testing is a technique based on the internal paths, structure, and implementation of the software under test.

- Structural testing is also known as white box testing.
- Structural testing obtains test requirements from implementation features.



Control-flow based criteria

- Criteria based on the flow of control within a program:
  - Statement coverage.
  - Decision coverage.
  - Condition coverage.
  - ...

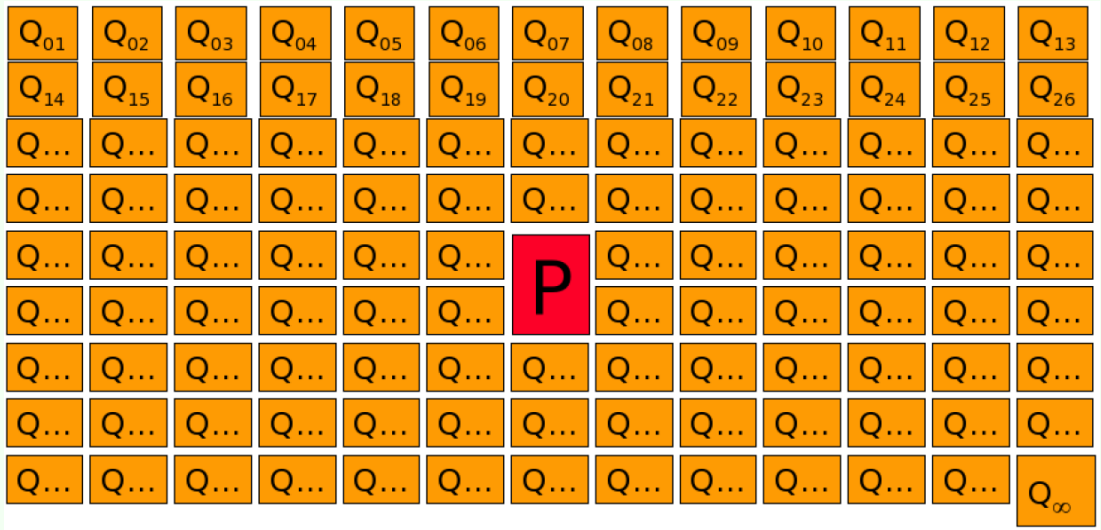
Data-flow based criteria

- Criteria based on the usage of data (variable creation, definition, and use):
  - All-uses.
  - All-potential-uses
  - ...

Example

Definition

Fault-based testing is a technique in which testing is based on historical information about common faults detected during the software development life cycle.



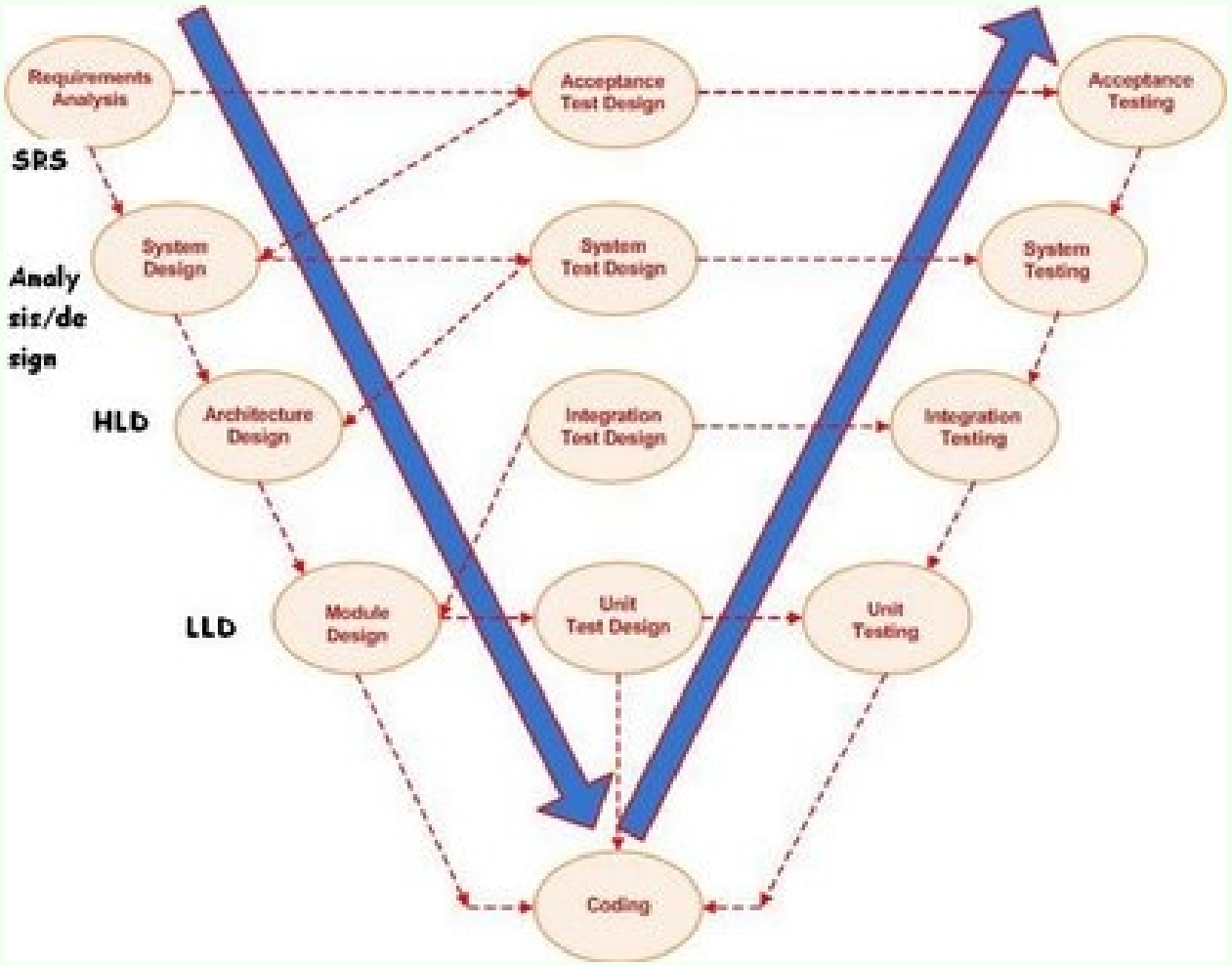
Fault-based test criteria

- Error seeding.
- Mutation:
  - Mutation analysis.
  - Interface mutation.
  - ...

Example

Software testing process

- When conducted in a systematic and clear-sighted way, software testing helps to:
  - Increase confidence that the product behaves according to its specification.
  - Highlight minimal characteristics of product quality.



References

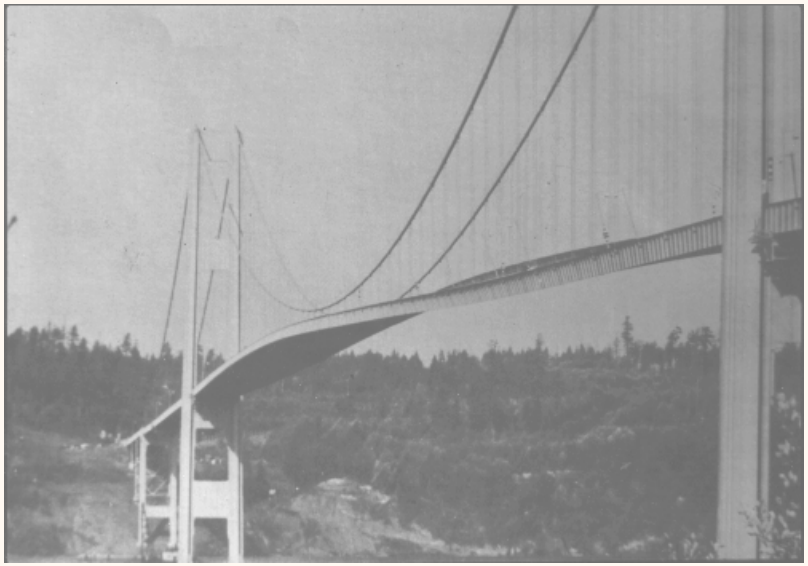
Credits

- Reviewers:
  - Fabiano Cutigi Ferrari
  - Otávio Augusto Lazzarini Lemos

Product failures

- The Tacoma Narrows Bridge is a pair of mile-long suspension bridges in the U.S. state of Washington, across the Tacoma Narrows, between Tacoma and the Kitsap Peninsula.
- It was opened to traffic on July 1, 1940.

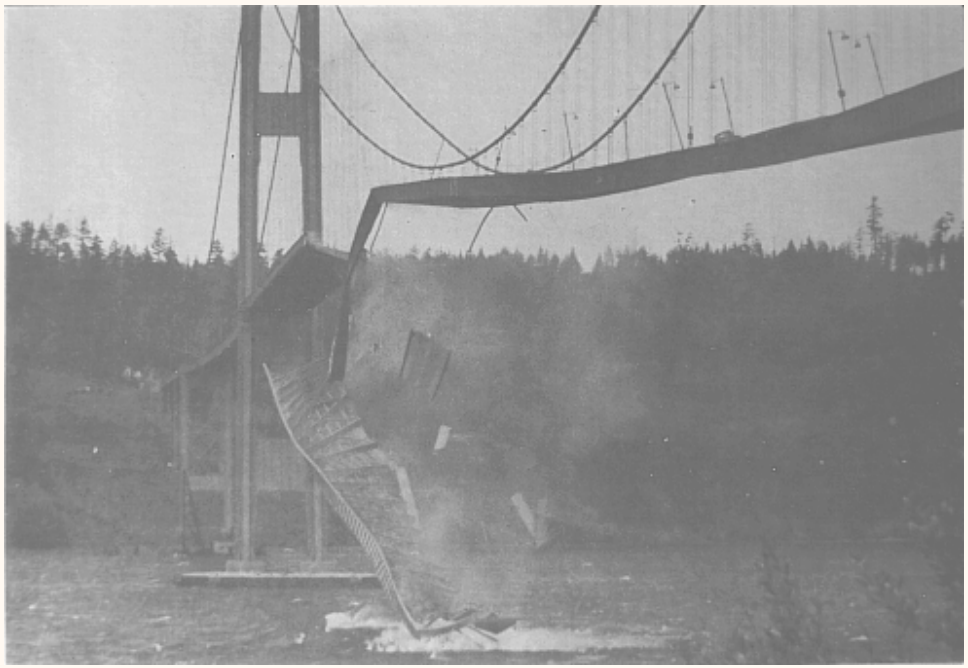
Before



Tacoma Narrows Bridge
Problem

- It collapsed four months later on November 7, 1940, at 11:00 AM (Pacific time) due to a physical phenomenon known as aeroelastic flutter caused by a 67 kilometers per hour (42 mph) wind.

After



Tacoma Narrows Bridge
Diagnostic

- Due to financing issues, shallower supports-girders 2.4 m deep were used.
  - This approach meant a slimmer, more elegant design and reduced construction costs compared to the original design.
- The decision to use shallow and narrow girders proved to be the first bridge's undoing.
  - With such girders, the roadbed was insufficiently rigid and was easily moved about by winds.
  - Bridge nicknamed as "Galloping Gertie".

- The failure of the bridge occurred when a never-before-seen twisting mode occurred, from winds at a mild 40 miles per hour (64 km/h).
- The amplitude of the motion produced by the fluttering increased beyond the strength of a vital part, in this case the suspender cables.
- Once several cables failed, the weight of the deck transferred to the adjacent cables that broke in turn until almost all of the central deck fell into the water below the span.

- Two solutions were devised:
  - Drill some holes in the lateral girders and along the deck so that the air flow could circulate through them (in this way reducing lift forces).
  - Give a more aerodynamic shape to the transverse section of the deck by adding fairings or deflector vanes along the deck, attached to the girder fascia.
- The second option was the chosen one; but it was not carried out, because the bridge collapsed five days after the solution was proposed.

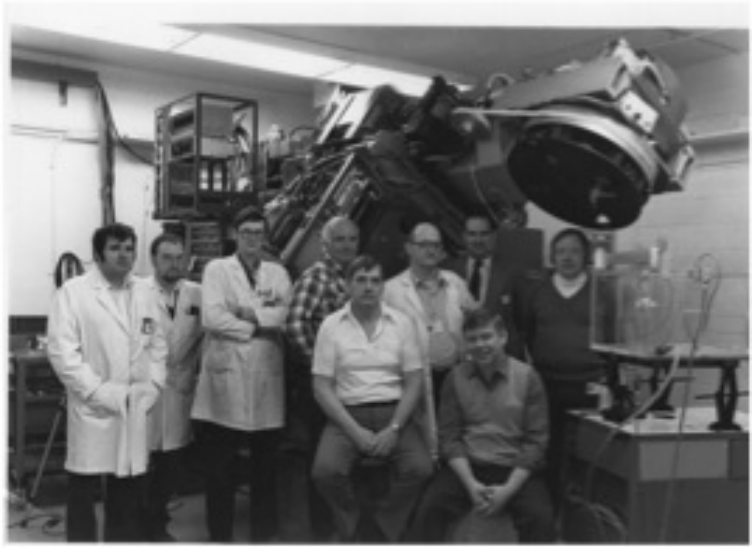
New York Subway

- In 1995, a train crashed into another train, killing the driver and hurting another 54 people.
- The distance between the signals (projected in 1918) was smaller than the distance required to stop the current trains (that are bigger, heavier and faster).
- The trains were updated without modification in the control system.
- Failure cause:
  - They updated part of the system, but did not validated the system as a whole.

Challenger mishap

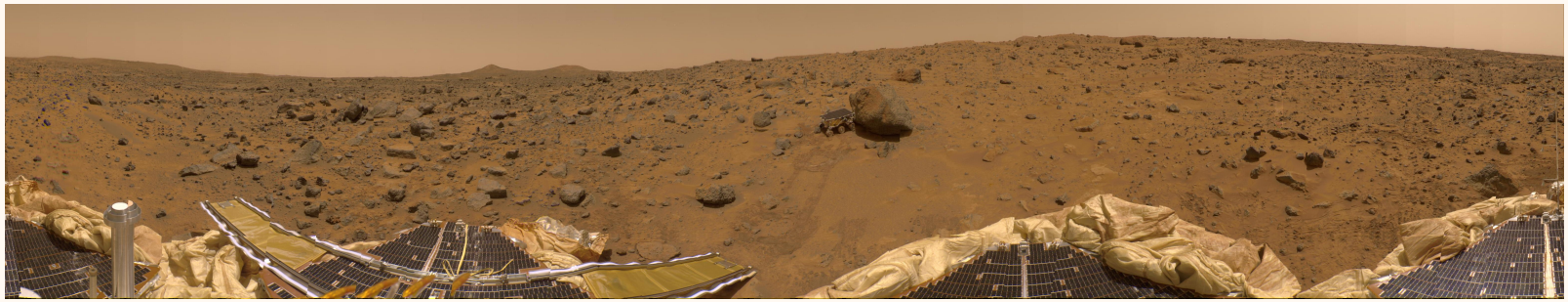
- In 1986, 73 seconds after taking off, a explosion destroyed the space shuttle Challenger, killing 7 astronauts.
- Investigations arrived at the conclusion that some rocket joints were not projected for the temperature and pressure they were suffering.
- Failure cause:
  - The pressure specification was not correct for the system requirement.
  - The tests were insufficient to detect the failure.

Software failures



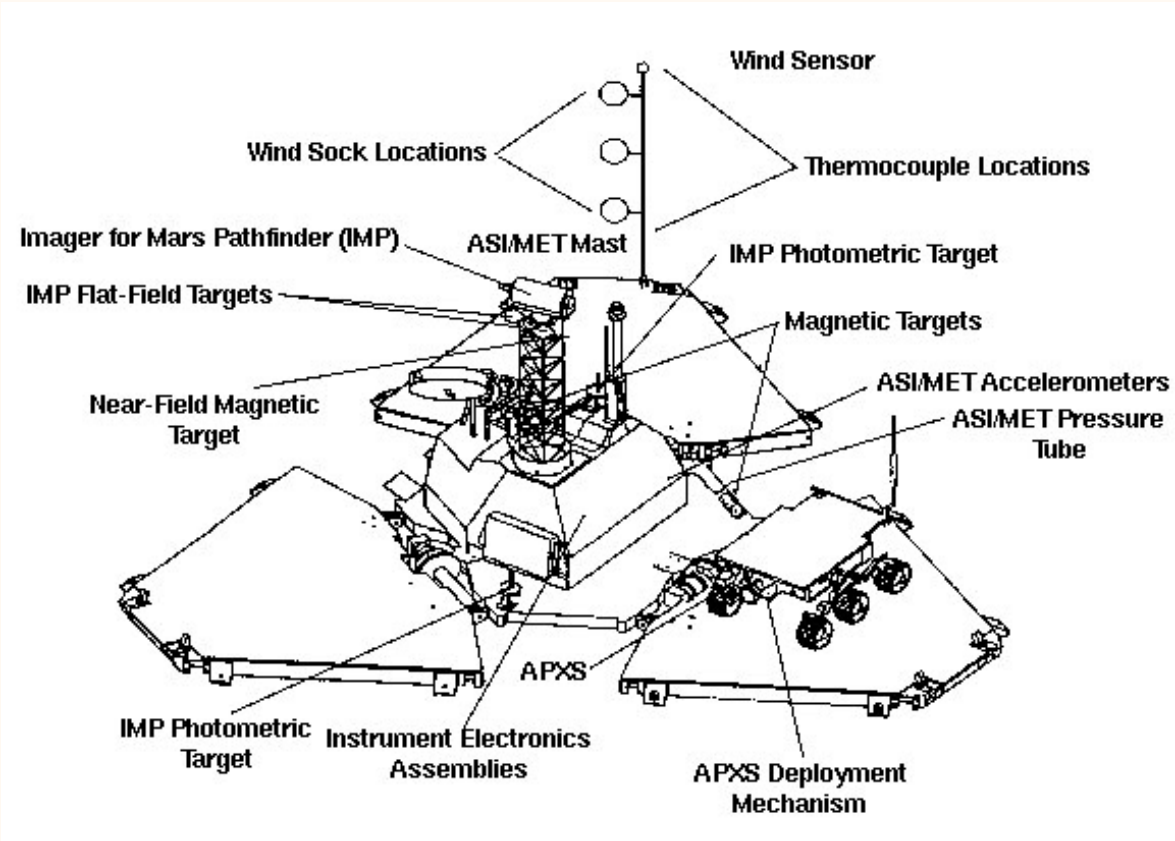
Mars Pathfinder
Description

- Mars Pathfinder was a low-cost planetary mission to Mars
  - Landed on Martian surface at July 4th, 1997.
- It demonstrated several new technologies for Mars exploration:
  - Airbag-based landing mechanism.
  - Use of rovers to collect and analyse soil and rock samples.
- The spacecraft had two parts: the lander and the rover.



Mars Pathfinder
Problem

- After collecting data for a long period, the lander would reset itself and all the data was lost.



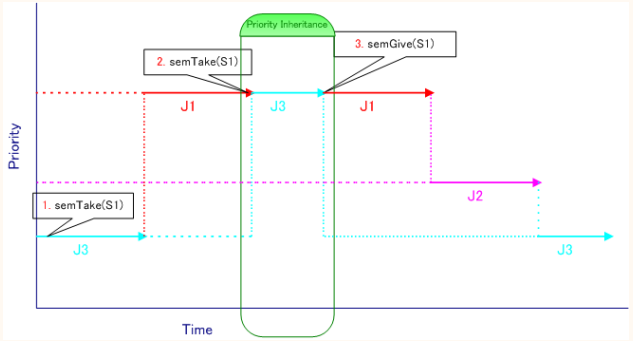
Mars Pathfinder
Diagnostic

- The lander software was concurrent and employed preemptive scheduling.
- Each thread had a priority and exchanged information between each other using an information bus.
  - Information bus = shared memory which access was controlled by a mutex.
- The information bus management system itself was a thread that run frequently, with a high priority.
- Another applications that run in the system was:
  - meteorological system, run much less frequently and with lower priority, and
  - communication thread: medium priority, but run frequently.

- This combination of threads **usually** worked correctly.
- However, the information bus management system could be blocked in the mutex in the following situation:
  1. Communication thread is scheduled and uses the processor, as it has higher priority than the meteorological thread.
  2. Communication thread can take as much time as it needs to run.
  3. However, a timer is expired whenever the information bus thread is not executed for a long time.
  4. The corrective measure taken by the timer is to reset the system.

- Change the value of a constant in the software, enabling priority inheritance.
  - When the information bus was blocked in the mutex, the meteorological thread would heritage the information bus thread priority, thus avoiding the communication thread to run for a too long time.

Priority inheritance example



Ariane 5
Description

- Ariane 5 is an expendable launch system used to deliver payloads into geostationary transfer orbit or low Earth orbit.
- The rocket took a decade to be developed and required 7 billions dollars.

Ariane 5
Problem

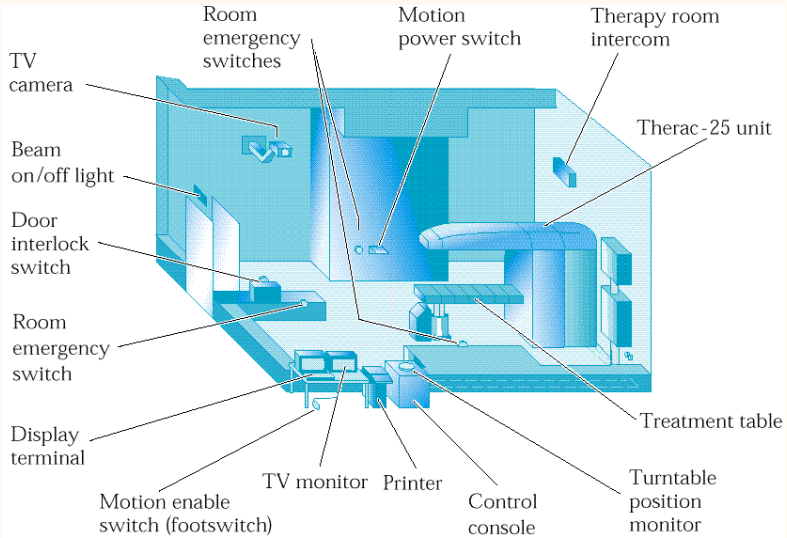
- On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure.
- Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded.

[hasprev=true,hasnext=false]

- The back-up Inertial Reference System failed, followed immediately by the failure of the active Inertial Reference System.
- The failure cause was a fault in the software that calculated the horizontal velocity of the rocket.
  - The variable that stored the value was 64 bit wide (floating point) and was incorrectly changed to 16 bits (signed integer).
  - The value was bigger than 32,767 (biggest value a signed integer can represent), causing a conversion failure.
- Insufficient testing for components reused from Ariane 4 were the cause of the failure.

Therac-25
Description

- Therac-25 was a computerized radiation therapy machine produced after the Therac-20 units.
- The machine offered two modes of radiation therapy:
  - direct electron-beam therapy,
  - megavolt X-ray therapy, which delivered X-rays produced by colliding high-energy (25 MeV) electrons into a target.



Therac-25
Description

- Therac-20 employed independent protective circuits and mechanical interlocks to protect against overdose.
- Therac-25 relied more heavily on software.
- FDA approved Therac-25 as pre-market equivalence to Therac-20 (even though the safety mechanisms were moved into the software, a major change from previous version of the machine.)

Therac-25
Problem

- The machine massively overdosed patients at least six times between June 1985 and January 1987.
  - Each overdose was several times the normal therapeutic dose.
  - The overdose resulted in the patient's severe injury or even death.

Therac-25
Diagnostic

- The accidents occurred when:
  1. **High-power** electron beam was activated instead of the intended **low power** beam, and
  2. beam spreader plate **not** rotated into place.
- The machine's software did not detect that this had occurred, and therefore did not prevent the patient from receiving a potentially lethal dose of radiation.

Therac-25
Diagnostic

- The failure only occurred when a particular nonstandard sequence of keystrokes was entered on the VT-100 terminal which controlled Therac-25.

PATIENT NAME : TEST
TREATMENT MODE : FIX BEAM TYPE: X ENERGY (MeV): 25

	ACTUAL	PRESCRIBED	
UNIT RATE/MINUTE	0	200	
MONITOR UNITS	50 50	200	
TIME (MIN)	0.27	1.00	
GANTRY ROTATION (DEG)	0.0	0	VERIFIED
COLLIMATOR ROTATION (DEG)	359.2	359	VERIFIED
COLLIMATOR X (CM)	14.2	14.3	VERIFIED
COLLIMATOR Y (CM)	27.2	27.3	VERIFIED
WEDGE NUMBER	1	1	VERIFIED
ACCESSORY NUMBER	0	0	VERIFIED

DATE : 84-OCT-26 SYSTEM : BEAM READY OP. MODE : TREAT AUTO
TIME : 12:55: 8 TREAT : TREAT PAUSE X-RAY 173777
OPR ID : T25V02-R03 REASON : OPERATOR COMMAND:

Therac-25
Diagnostic

- The primary reason should be attributed to the bad software design and development practices and not explicitly to several coding errors that were found.
  - The equipment control task did not properly synchronize with the operator interface task, so that race conditions occurred if the operator changed the setup too quickly.
  - AECL had never tested the Therac-25 with the combination of software and hardware until it was assembled at the hospital.

Therac-25
Solution

- First suggested solution (proposed by the manufacturer):
  - Change in operating procedures:
    - The key used for moving the cursor back through the prescription sequence (i.e., cursor "UP" inscribed with an upward pointing arrow) must not be used for editing or any other purpose.
    - To avoid accidental use of this key, the key cap must be removed and the switch contacts fixed in the open position with electrical tape or other insulating material.
- After several years of troubleshooting, the final solution was comprised of numerous software fixes, the installation of independent, mechanical safety interlocks, and a variety of other safety related changes.

Thunder Horse
Description

- Thunder Horse is a semi-submersible deep-water platform:
  - Displacement of about 130,000 tons.
  - Largest and, reportedly, the most technologically advance deep-water platform ever built.

Before

Thunder Horse
Problem

- In mid July 2005, Hurricane Dennis swirled in the Gulf of Mexico.
- The platform was evacuated as a precaution.
- Upon return to the platform on July 12 2005, it was found precariously listing 20 to 30 degrees.
  - The lower deck of the platform was at sea level.

After

Thunder Horse
Diagnostic

- Thunder Horse oilfield listing after an hurricane due to a ballast system error.

La Tosca at San Diego
Description

- The opera La Tosca (Giacomo Puccini) debuted just over one hundred years ago, at the Teatro Constanzi in Rome on January 14, 1900.
- Soon after its premiere, it became one of the most popular operas in the repertoire, and it remains so to this day.
- It was the candelabra that played a prominent role in a San Diego performance of Tosca in 1956.
  - The script called for Tosca to blow out the four candles in the candelabra before dramatically placing a candle on either side of Scarpia and a crucifix on his breast and exiting the stage.

La Tosca at San Diego
Problem

- In San Diego, the candles were electric, and the order of their going out was fixed on a computer tape along with all the rest of the lighting cues.
- The tape obeyed the stage manager's signal and snuffed the candles exactly as Tosca blew them out.
  - Except that, on this occasion, the programming was wrong and it blew them out in a different order from hers.
  - She blew to the right, the candle on the left went out, she blew the back one, the one in front went out!

Software requirements specification

The software read three values that represents the edges of a triangle. For those values, the software should write if the triangle is isosceles, scalene, or equilateral.

Your task

- You were hired to develop this software. How would you do it? Think about:
  - design,
  - implementation,
  - testing.

My task

- I was the guy that hired you to implement software.
- As soon as you deliver the first release of the software, I came up with several issues.

Issues

- Why the triangle (5, 5, 3) was not recognized as isosceles?
- Why did the software accepted triangles which the sum of two edges is less than the third edge?
- Why did the software accepted triangles which edges are zero?
- Why did the software accepted triangles which edges are negative?
- Why did the software accepted triangles which edges are not even a number?

Diagnostic

- The software requirement specification was far from complete (many implicit requirements).
- Nonetheless, several errors could be caught it test cases were designed for the software **before** delivering or even implementing it.

Did you tested for. . .

- a valid scalene triangle?
- a valid isosceles triangle?
- a valid equilateral triangle?
- valid isosceles triangles (with permutated edges)?
- a value of zero?
- negative values?
- the sum of two edges is equal to the third edge?
- the sum of two edges is less than the third edge?
- all edges with a value of zero?
- non-integer values?
- insufficient values (e.g., 2 instead of 3 edges)?
- the correct triangle type for the specified input in all the previous tests?

Incorrect statement

- Consider the following statement, as defined in the software requirements specification.
  - The z dimension value of the vortex is the sum of its x and y dimension values.
- By an unknown reason (the cat walked on the keyboard, insomnia, keyboard malfunctioning), the programmer wrote a statement that does not correctly implements the requirement:

```
z = x - y.
```

- This programmer incurred into a mistake.

Incorrect statement
Description

- Consider the following software requirement: "The value of `y` should be a function of `x` and `y`, as defined by the expression `y = (x - z) / 2`".
- The programmer, when implementing the requirement, wrote:

```
y = x - z / 2
```

Incorrect statement
Diagnostic

- The programmer wrote an incorrect data definition in the software! That's a fault!
- For `x = 10` and `z = 8`, the expected output is `y = 1`, but the incorrect version's output is `y = 6`.

Incorrect statement

- Consider the following statement, as defined in the software requirements specification.
  - The z dimension value of the vortex is the sum of its x and y dimension values.
- By an unknown reason (the cat walked on the keyboard, insomnia, keyboard malfunctioning), the programmer wrote a statement that does not correctly implements the requirement:

```
z = x - y.
```

- This programmer incurred into a mistake, which characterized a fault.

Incorrect statement

- If such a fault is activated (executed) with `x = 0`, regardless of the value of `y`, no incorrect output is produced.
  - Although the fault is activated, it does not lead to an error and no failure occurs.
- For any other value different from `x = 0`, the fault activation causes an error on the variable `z`.
  - Such an error, when propagated until the product output, will result in a failure.

Physician analogy for defect taxonomy

- Consider a physician making a diagnosis for a patient.
- The patient enters the physician's office with a list of **failures** (that is, symptoms).
- The physician then must discover the **fault**, or root cause of the symptom.
- To aid in the diagnosis, the physician may order tests that look for anomalous internal conditions, such as high blood pressure, an irregular heartbeat, high levels of blood glucose, or high cholesterol.
  - In our terminology, these anomalous internal conditions correspond to **errors**.

numZero
Description

- Is there any fault in this program?

```
class numZero
{
    /**
     * If arr is null throw NullPointerException, else
     * return the number of occurrences of zero in arr.
     */
    public static int numZero (int[] arr)
    {
        int count = 0;
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] == 0) {
                count++;
            }
        }
        return count;
    }
}
```

numZero
Diagnostic

- The fault in this program is that it starts looking for zeroes at index 1 instead of index 0, as is necessary for arrays in Java.
  - For example, numZero([2, 7, 0]) correctly evaluates to 1, while numZero([0, 7, 2]) incorrectly evaluates to 0.
    - In both of these cases the fault is executed.
    - Although both of these cases result in an error, only the second case results in failure.

```
for (int i = 1; i < arr.length; i++) {
    if (arr[i] == 0) {
        count++;
    }
}
return count;
```

- For the first example given above, the state at the if statement on the very first iteration of the loop is (x = [2, 7, 0], count = 0, i = 1, PC = if).
- This state is in error precisely because the value of i should be zero on the first iteration.
- However, since the value of count is coincidentally correct, the error state does not propagate to the output, and hence the software does not fail.

```
for (int i = 1; i < arr.length; i++) {
    if (arr[i] == 0) {
        count++;
    }
}
return count;
```

- For the second example given above, the corresponding error state is (x = [0, 7, 2], count = 0, i = 1, PC = if).
- In this case, the error propagates to the variable count and is present in the return value of the method. Hence a failure results.

```
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] == 0) {
                count++;
            }
        }
        return count;
```

Sort test cases

- Consider a sort implementation for a array of integers.
- The input of the sort application is an array, set as the argument of the application, and the result is printed to the console.
- For example, for the input array '3 1 7 4', the result would be:

```
$ sort 3 1 7 4
1 3 7 4
```

Sort test cases

- A possible set of test cases is:
  - $< [3, 1, 7, 4], [1, 3, 7, 4] >$
  - $< [1, 2, 3, 4], [1, 2, 3, 4] >$
  - $< [0, 1, 0, 2], [0, 0, 1, 2] >$
  - $< [], [] >$
  - $< [a, b, 0, 1], errormessageprintedtoconsole >$

Sick person visiting a doctor

- Consider the analogy of a person visiting a doctor because of an overall feeling of malaise.
- If the doctor runs some laboratory tests that do not locate the problem, we do not call the laboratory test "successful".
  - They were unsuccessful tests in that the patient's net worth has been reduced by the expensive laboratory fees, and the patient is still ill.
- However, if a laboratory test determines that the patient has a peptic ulcer, the test is successful.
  - The doctor can now begin the appropriate treatment.

Analogy

- The sick person should be imagined as the software.
- The laboratory tests are the test cases designed for the software.
- Peptic ulcer is the failure found by the test.

numZero test cases
Description

- Consider the following code:

```
public static int numZero(int[] arr) {
    int count = 0;
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] == 0) {
            count++;
        }
    }
    return count;
}
```

- Some test cases that could be designed for this code are:
  - ([2, 7, 0], 1)
  - ([0, 7, 2], 1)
  - ([], 0)
  - ([0, 0, 0], 3)

Cascading test cases

- The first test case exercises a particular feature of the software and then leaves the system in a state such that the second test case can be executed.
- While testing a database consider these test cases:
  1. create a record,
  2. read the record,
  3. update the record,
  4. read the record,
  5. delete the record, and
  6. read the deleted record.

Kiddie oracle example

- Consider the factorial of a number.
- For a given input number, if the result looks correct, then it is correct.
- For small input numbers, like 2, 3, it is easy to verify that:
  - $2! = 2$
  - $3! = 6$
- But what about bigger numbers?
  - $11! = 39916800$
  - $21! = 51090942171709440000$
- Actually, $21! = 51090\mathbf{9}42171709440000$ (instead of $51090\mathbf{8}42171709440000$)

Regression test suite oracle example

- Mozilla runs build computers that continually build the latest source code.
- A tool, called Thinderbox, collects the test results and compare to the previous version of the product under testing.
- It uses tables to describe status of the source tree for the platform, product, and code branch:
  - 1
  - The green bar means the latest code compiles and passes the tests that are run on the box.
  - Red means the build failed during compilation.
  - Orange means the binary was built successfully, but failed some of the tests.
  - Yellow means the build and testing are in progress.
- Tinderbox for Firefox:
  - `http://tinderbox.mozilla.org/showbuilds.cgi?tree=Firefox`

- Any Java implementation (Java Virtual Machine and standard libraries) can be tested against the Java Compatibility Test Tools (Java CTT), available at `http://www.jcp.org/en/resources/tdk`.
- However, the Java CTT is not freely available (it requires you to be JCP member).
- As an alternative, you can use the Mauve, which is available at `http://sources.redhat.com/mauve/`.

Pentium FDIV bug
Description

- In 1994, Intel introduced its Pentium microprocessor, and a few months later, a mathematician found that the chip gave incorrect answers to certain floating-point division calculations.
  - The chip was slightly inaccurate for a few pairs of numbers.
- Example: A number multiplied and then divided by the same number should result in the original number)
  - Expected result: $4195835 * 3145727/3145727 = 4195835$
  - Flawed Pentium result: $4195835 * 3145727/3145727 = 4195579$

- The fault was the omission of five entries in a table of 1,066 values (part of the chip's circuitry) used by a division algorithm.
  - The five entries should have contained the constant $+2$, but the entries were not initialized and contained zero instead.

Pentium FDIV bug
Solution

- The mistake that caused the fault was very difficult to find during system testing.
  - Intel claimed to have run millions of test cases using this table.
- The table entries were left empty because a loop termination condition was incorrect
  - The loop stopped storing numbers before it was finished.
- This turns out to be a very simple fault to find during unit testing
  - Analysis showed that almost any unit level coverage criterion would have found this multi-million dollar mistake.

Mockito

- Mockito is a mocking framework.
  - Mock is something that mimics an unit of software.
  - A stub mocks an unit of software.

How does it work

- A mock is created for a given class.
  - This mock is an instance of the given class, but which behaviour is undefined.
- After the instantiation of the mock, the behaviour of every method that will be tested must be defined.
  - For instance, you can define that the method `get`, when using the parameter 35, will return the string `Tomato`.
- Thus, complex objects do not need to be implemented: just stub the methods used in the test case.

Test case

- Imagine the following situation. The unit under testing is the class `IdentityMethodSet`.
  - This class defines a set that uses as identity the method which name is given as parameter in the constructor (which is `toString` in the example).
- You need to test the behaviour of the `IdentityMethodSet` for a class that can throw an exception when the method assigned as identity is used.
  - `IdentityMethodSet` must throw an `UnsupportedOperationException` if it fails to use the identity method of the object.

Test case

- However, it may be difficult to find a class that implements such behaviour.
- So, instead of finding a class that has such behaviour, we just create a mock that stubs the method `toString`.
  - And we configure this method, in the mock, to throw an exception.

Implementation

```
import static org.mockito.Mockito.*;

public class IdentityMethodSetTest {
{
    @Test(expected=UnsupportedOperationException.class)
    public void testNew_InvalidIdentityMethod_UnsupportedOp
    {
        InetAddress clazz = mock(InetAddress.class);
        when(clazz.toString()).thenThrow(new NullPointerExce
        IdentityMethodSet set = new IdentityMethodSet<String
        set.add(clazz);
    }
}
```

Implementation details

- The mock is created using the method `mock` (imported from `org.mockito.Mockito`).

Mars Climate Orbiter

- Mars Climate Orbiter was a NASA spacecraft that would study the Martian weather, climate, water and carbon dioxide budget.
- It was intended to enter orbit at an altitude of 140.5 - 150 km above Mars.
- However, a navigation error caused the spacecraft to reach as low as 57 km.
- The spacecraft was destroyed by atmospheric stresses and friction at this low altitude.

Mars Climate Orbiter
Diagnostic

- The navigation error arose because of a software programming error.
  - The thruster control software did not properly interpret the data fed to it.
- The modules of the thruster control were created by separate software groups.
- One module computed thruster data in English units and forwarded the data to a module that expected data in the International System units (meters).
- This is a very typical integration fault (but in this case enormously expensive, both in terms of money and prestige).

Ghost Train
Description

- In 1995, failures were registered by the sensors in the rails situated in tunnels within England channels.
  - The trains stopped due to suspicion that another train was in the rail.
- The cause was that the salty water fog was confusing the sensors.
  - The system requirements probably didn't considered the salty water fog as a possible event.
  - Adequate system testing and validation could have detected the problem.

Test criterion example

- Suppose we are given the enviable task of testing bags of jelly beans. We need to come up with ways to sample from the bags.
- Suppose these jelly beans have the following six flavors and come in four colors: Lemon (colored Yellow), Pistachio (Green), Cantaloupe (Orange), Pear (White), Tangerine (also Orange), and Apricot (also Yellow).
- A simple approach to testing might be to test one jelly bean of each flavor. Then we have six test requirements, one for each flavor.
- We satisfy the test requirement "Lemon" by selecting and, of course, tasting a Lemon jelly bean from a bag of jelly beans.

Test criterion example

- The "flavor criterion" yields a simple strategy for selecting jelly beans.
- In this case, the set of test requirements, TR, can be formally written out as $TR = \{flavor = Lemon, flavor = Pistachio, flavor = Cantaloupe, flavor = Pear, flavor = Tangerine, flavor = Apricot\}$.

- Consider the function `blech`, implemented as follows:

```
int blech(int j) {
    j = j - 1; // should be j = j + 1;
    j = j / 30000;
    return j;
}
```

- Input domain:
  - Consider an integer type of 16 bits (2 bytes).
  - The lowest possible input value is -32,768 and the highest is 32,767.
  - Thus there are 65,536 possible inputs into this small software.
- Which test cases can detect the fault?

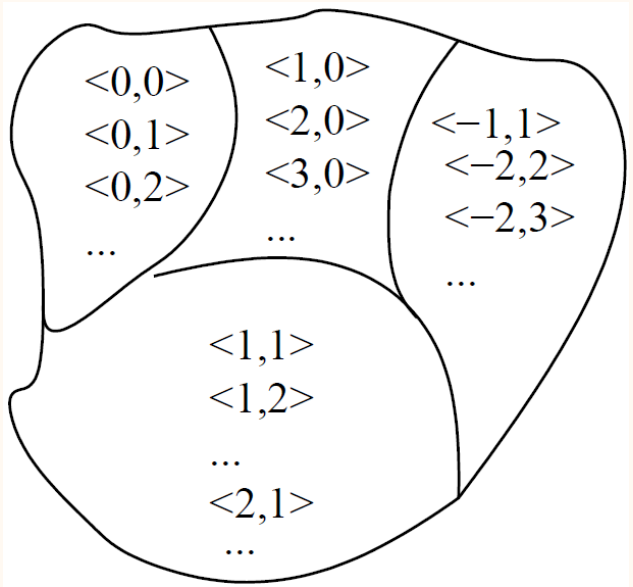
- Only four out of the possible 65,536 input values will find this fault:

Test Cases Input(j)	Expected Output	Actual Result
-30000	0	-1
-29999	0	-1
30000	1	0
29999	1	0

```
int blech(int j) {
    j = j - 1; // should be j = j + 1;
    j = j / 30000;
    return j;
}
```

Functional testing example

Consider the function x^y, where x is an integer and y is a non-negative integer. The input domain is: for every tuple (x,y), consider all possible values of x and $y >= 0$. The input domain can be partitioned as follows:



Structural testing example

Example

Consider the following source code snippet [?, p. 45].

Source code

```
public int foo(int a, int b, int x) {
    if (a > 1 && b == 0) {
        x = x/a;
    }
    if (a == 2 || x > 1) {
        x = x + 1;
    }
    return x;
}
```

Structural testing example

Source code

```
public int foo(int a, int b, int x) {
    if (a > 1 && b == 0) {
        x = x/a;
    }
    if (a == 2 || x > 1) {
        x = x + 1;
    }
    return x;
}
```

Statement criterion

- Test cases that cover all test requirements for **statement criterion**:
  - $(a = 2, b = 0, x = 3)$
- But what if the error is related to the first decision?
  - Instead of an *and*, it should be an *or*.

Structural testing example

Source code

```
public int foo(int a, int b, int x) {
    if (a > 1 && b == 0) {
        x = x/a;
    }
    if (a == 2 || x > 1) {
        x = x + 1;
    }
    return x;
}
```

Decision criterion

- Test cases that cover all test requirements for **decision criterion**:
  - $(a = 3, b = 0, x = 3)$
  - $(a = 2, b = 1, x = 1)$

Example

Consider the following source code snippet [?, p. 45].

Source code

```
public void foo(int a, int b, int x) {
    if (a > 1 && b == 0) {
        x = x/a;
    }
    if (a == 2 || x > 1) {
        x = x + 1;
    }
}
```

Mutation analysis

- Mutation analysis performs single changes in the source code.
- For example, comparators such as == can be changed to !=.
- The following test case will reveal the error in the mutant:
  - $a = 3, b = 0, x = 3$

Original application

```
public int foo(int a, int b, int x) {
    if (a > 1 && b == 0) {
        x = x/a;
    }
    if (a == 2 || x > 1) {
        x = x + 1;
    }
    return x;
}
```

Mutation analysis

- Mutation analysis performs single changes in the source code.
- For example, comparators such as == can be changed to !=.
- The following test case will reveal the error in the mutant:
  - $a = 3, b = 0, x = 3$

Mutant

```
public int foo(int a, int b, int x) {
    if (a > 1 && b != 0) {
        x = x/a;
    }
    if (a == 2 || x > 1) {
        x = x + 1;
    }
    return x;
}
```