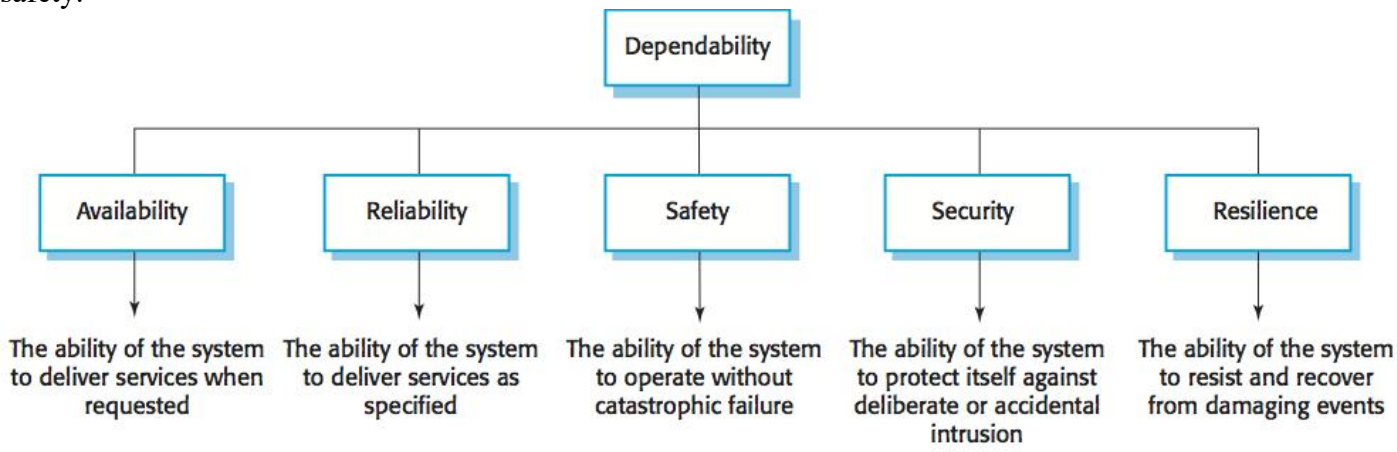
PRINCIPAL PROPERTIES OF DEPENDABILITY

Availability: The probability that the system will be up and running and able to deliver useful services to users.
Reliability: The probability that the system will correctly deliver services as expected by users.
Safety: A judgment of how likely it is that the system will cause damage to people or its environment.
Security: A judgment of how likely it is that the system can resist accidental or deliberate intrusions.
Resilience: A judgment of how well a system can maintain the continuity of its critical services in the presence of disruptive events such as equipment failure and cyberattacks.

Other properties of software dependability:
Repairability reflects the extent to which the system can be repaired in the event of a failure;
Maintainability reflects the extent to which the system can be adapted to new requirements;
Survivability reflects the extent to which the system can deliver services whilst under hostile attack;

Error tolerance reflects the extent to which user input errors can be avoided and tolerated.
Many dependability attributes depend on one another. Safe system operation depends on the system being available and operating reliably. A system may be unreliable because its data has been corrupted by an external attack. Denial of service attacks on a system are intended to make it unavailable. If a system is infected with a virus, you cannot be confident in its reliability or
safety.



How to achieve dependability?

- Avoid the introduction of accidental errors when developing the system.
- Design V & V processes that are effective in discovering residual errors in the system.
- Design systems to be fault tolerant so that they can continue in operation when faults occur.
- Design protection mechanisms that guard against external attacks.
- Configure the system correctly for its operating environment.
- Include system capabilities to recognize and resist cyberattacks.
- Include recovery mechanisms to help restore normal system service after a failure.

Dependability costs tend to increase exponentially as increasing levels of dependability are required because of two reasons. The use of more expensive development techniques and hardware that are required to achieve the higher levels of dependability. The increased testing and system validation that is required to convince the system client and regulators that the required levels of dependability have been achieved.

FUNCTIONAL PROGRAM TESTING

Identify Independently Testable Features

- Functional specifications can be large and complex so they should be decomposed , so that we can devise separate test cases for each functionality
- The preliminary step of functional testing consists in partitioning the specifications into features that can be tested separately.
- Independently testable features are described by identifying all the inputs that form their execution environments.

Identify Representative Classes of Values or Derive a Model

- The next step of a testing process consists of identifying which values of each input should be selected to form test cases.
- Representative values can be identified directly from informal specifications expressed in natural language. Or representative values may be selected in- directly through a model

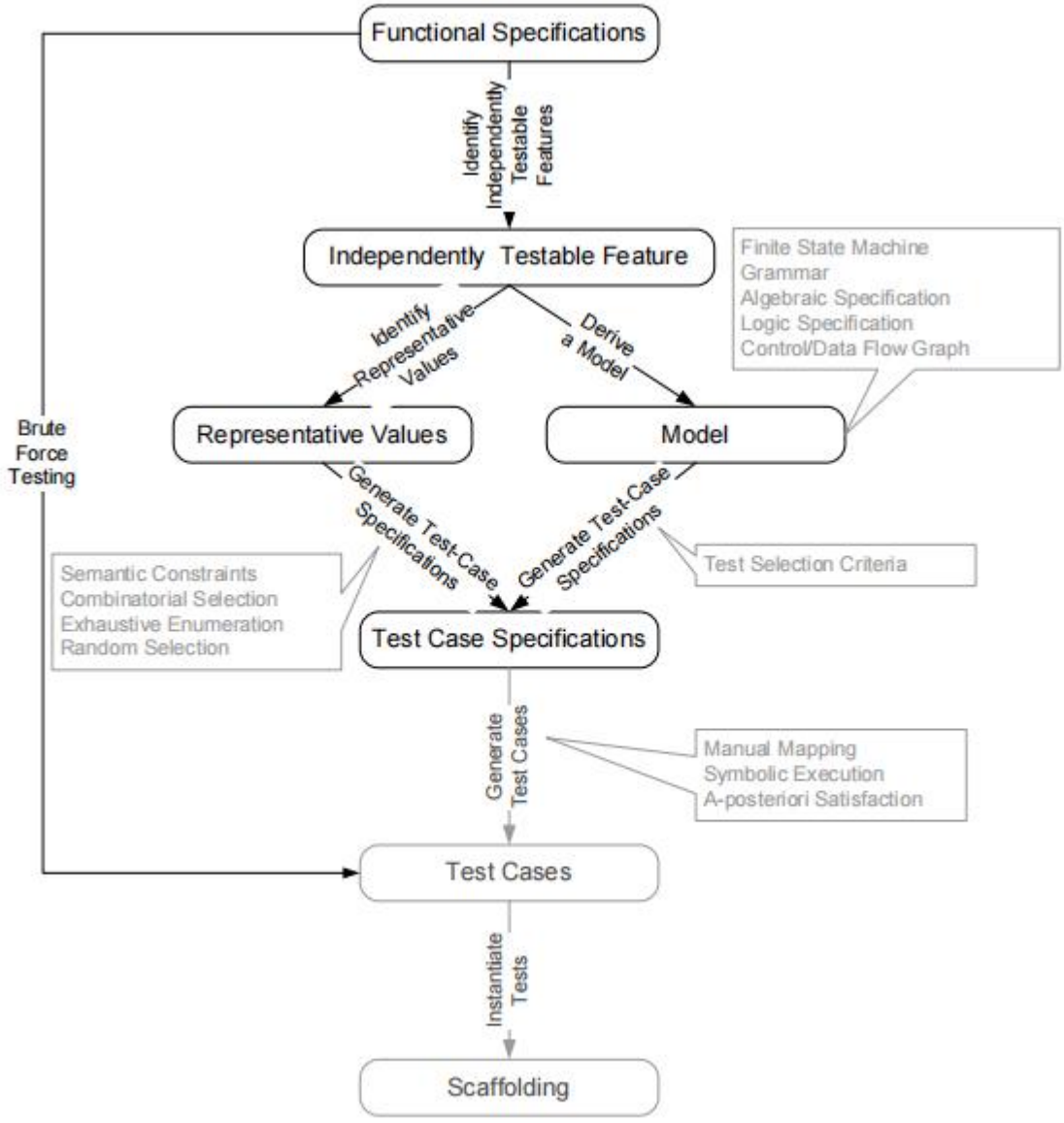


Figure 10.3: The main steps of a systematic approach to functional program testing.

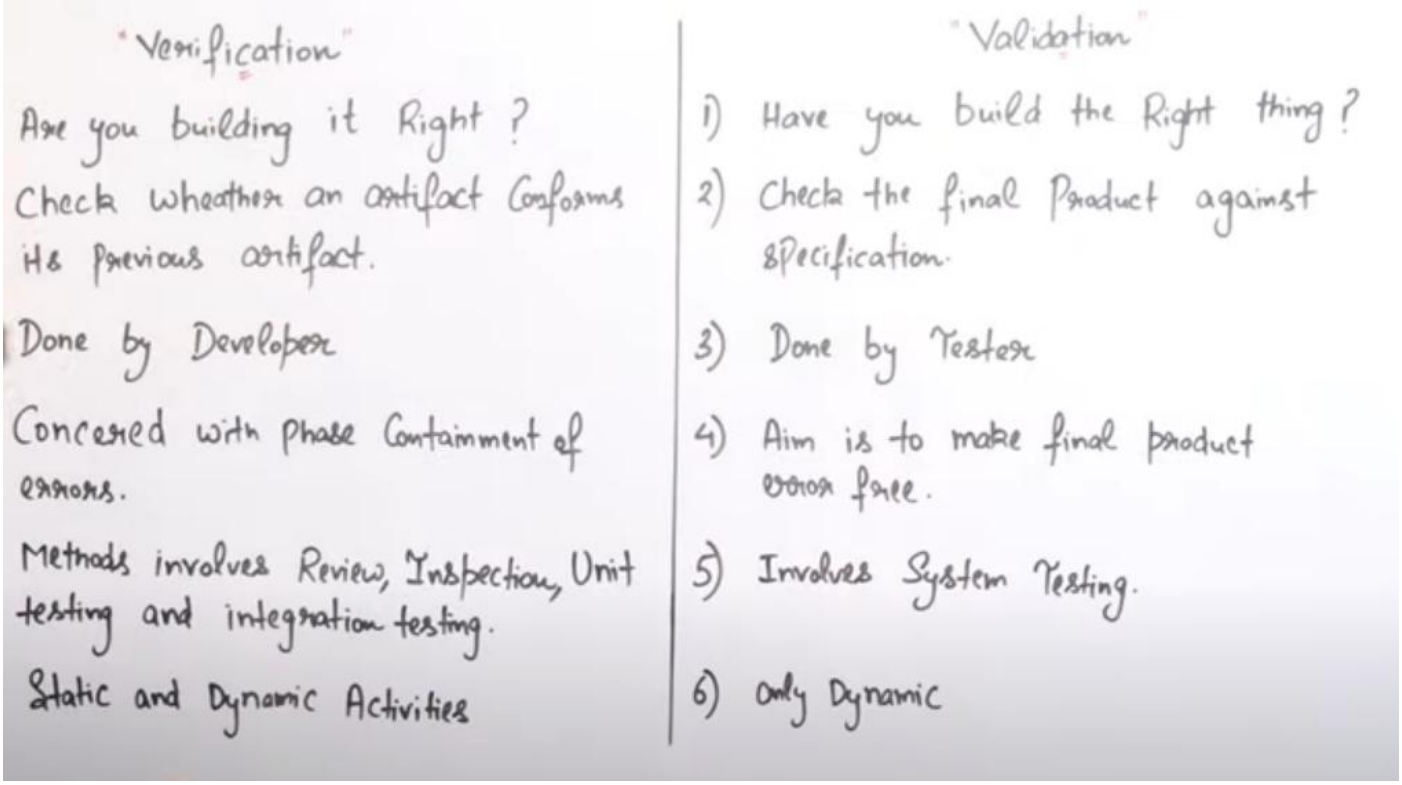
Generate Test Case Specifications

- Test specifications are obtained by suitably combining values for all inputs of the functional unit under test.
- If representative values were explicitly enumerated in the previous step, then test case specifications will be elements of the Cartesian product of values selected for each input.
- If a formal model was produced, then test case specifications will be specific behaviors or combinations of parameters of the model, and a single test case specification could be satisfied by many different concrete inputs.

Generate Test Cases and Instantiate Tests

- The test generation process is completed by turning test case specifications into test cases and instantiating them.
- Test case specifications can be turned into test cases by selecting one or more test cases for each test case specification.
- Test cases are implemented by creating the scaffolding required for their execution.

VERIFICATION VS VALIDATION



Verification	Validation
We check whether we are developing the right product or not.	We check whether the developed product is right.
Verification is also known as static testing.	Validation is also known as dynamic testing.
Verification includes different methods like Inspections, Reviews, and Walkthrough.	Validation includes testing like functional testing, system testing, integration, and User acceptance testing.

It is a process of checking the work-products (not the final product) of a development cycle to decide whether the product meets the specified requirements.	It is a process of checking the software during or at the end of the development cycle to decide whether the software follow the specified business requirements.
Quality assurance comes under verification testing.	Quality control comes under validation testing.
The execution of code does not happen in the verification testing.	In validation testing, the execution of code happens.
In verification testing, we can find the bugs early in the development phase of the product.	In the validation testing, we can find those bugs, which are not caught in the verification process.
Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements.	Validation testing is executed by the testing team to test the application.
Verification is done before the validation testing.	After verification testing, validation testing takes place.
In this type of testing, we can verify that the inputs follow the outputs or not.	In this type of testing, we can validate that the user accepts the product or not.

V&V PROCESS START AND END

- Verification and validation start as soon as we decide to build a software product, or even before.
- Execution of test cases is a small part of verification and validation(V&V) process required to maintain a quality of software product
- Validation is an activity that ensures that an end product meets the needs & expectation of users
- Verification takes place while the product is still under development,verification is done before validation
- Verification is executed by the quality assurance team/development team to make sure product has been developed right.
- Validation is performed upon the completion of given module or an entire application
- V&V activities continue through each small or large change to the system.

QUALITY OF SUCCESSIVE RELEASES

- Software test and analysis does not stop at the first release.software products operate for many years and undergo many changes
- They adapt to environment changes.for example ,introduction of new device drivers evolution of the operating system and changes in the underlying database
- They also evolve to serve new and changing user requirement
- Ongoing quality tasks include test and analysis of new and modified code,re execution of system tests and extensive record keeping
- Major revisions-->point releases
- Smaller revisions-->patch level
- The full quality process is repeated in miniature for each point release, including everything from inspection of revised requirements to design and execution of new unit, integration, system, and acceptance test cases. A major point release is likely even to repeat a period of beta testing.
- Patch level revisions are often urgent for at least some customers.Test and analysis for patch level revisions is abbreviated, and automation is particularly important for obtaining a reasonable level of assurance with very fast turnaround.Here regression test are conducted for fault.

SOFTWARE TESTING

It is a process of evaluating and verifying that a software product or application does what it is supposed to do.It is a process used to identify the correctness,completeness and quality of the developed software.

VERIFICATION

"Verifification" is checking the consistency of an implementation with a specification. Here, "specification" and "implementation" are roles, not particular artifacts.

VALIDATION

Assessing the degree to which a software system actually fulfills its requirements, in the sense of meeting the user's real needs, is called validation.

TEST CASES

A test case includes not only input data but also any relevant execution conditions and procedures and a way of determining whether the program has passed or failed the test on a particular execution.

SENSITIVITY PRINCIPLE

It is better to fail every time than sometime

ATTRIBUTES OF GOOD SOFTWARE

Dependability· Reliability·Scalability· Usability· Portability · Efficiency ·Correctness · Integrity · Adaptability · Accuracy · Robustness.

ALPHA TESTING

The approach to verifying reliability, using a sample of real users is known as alpha testing if the tests are performed by users in a controlled environment.

BETA TESTING

The approach to verifying reliability, using a sample of real users is known as beta testing if the tests are performed in their own environment without interference or close monitoring.

REGRESSIVE TESTING

Regression testing is a software testing practice that ensures an application still functions as expected after any code changes, updates, or improvements.

PRINCIPLE OF ANALYSIS AND TESTING

Sensitivity:-

- Better to fail every time than sometimes
- A test selection criterion works better if every selected test provides the same result, i.e., if the program fails with one of the selected tests(one set of inputs), it fails with all of them (reliable criteria)
- Run time deadlock analysis works better if it is machine independent, i.e., if the program deadlocks when analyzed on one machine, it deadlocks on every machine

Redundancy:-

- Redundant checks can increase the capabilities of catching specific faults early or more efficiently.
- Making intentions explicit

- Static type checking is redundant with respect to dynamic type checking, but it can reveal many type mismatches earlier and more efficiently.
- Validation of requirement specifications is redundant with respect to validation of the final software, but can reveal errors earlier and more efficiently.

Partition:-
- Divide and conquer
- It is a general engineering principle.
- Dividing complex problem into sub problems to be solved
- Independently is probably the most common human problem solving Strategy.
- Partitioning can be applied both at process and technique levels. At the Process level we divide the complex activity into simple activities.
- For example, testing is usually divided into unit, integration, sub system and system testing
- At the analysis level first construct a model of the system and then analysis, in this way divide the analysis into two sub tasks.

Restriction:-
- Making the problem easier
- Suitable restrictions can reduce hard (unsolvable) problems to simpler (solvable) problems.
- A weaker spec may be easier to check: it is impossible (in general) to show that pointers are used correctly, but the simple Java requirement that pointers are initialized before use is simple to enforce.

- A stronger spec may be easier to check: it is impossible (in general) to show that type errors do not occur at run-time in a dynamically typed language, but statically typed languages impose stronger restrictions that are easily checkable.

Visibility:-
- Judging status
- The ability to measure progress or status against goals
- X visibility = ability to judge how we are doing on X
- Schedule visibility = “Are we ahead or behind schedule,”
- Quality visibility = “Does quality meet our objectives?”
- The visibility is closely related to observability to extract useful information from the software artifact.

Feedback:-
- Tuning the development process.
- Learning from experience: Each project provides information to improve the next.

ADEQUACY CRIETERIA

- We will use the term test obligation for test specifications imposed by adequacy criteria,a test suite satisfies an adequacy criterion if all the tests succeed and if every test obligation in the criterion is satisfied by at least one of the test cases in the test suite. For example, the statement coverage adequacy criterion is satisfied by a particular test suite for a particular program if each executable statement in the program (i.e., excluding comments and declarations) is executed by at least one test case in the test suite.
- A fault-based adequacy criterion that seeds a certain set of faults would be satisfied if, for each of the seeded faults, there is a test case that passes for the original program but fails for the program with (only) that seeded fault.
- It is quite possible that no test suite will satisfy a particular test adequacy criterion for a particular program.
- One approach to overcoming the problem of unsatisfyable test obligations is to simply exclude any unsatisfiable obligation from a criterion
- If the number of infeasible test obligations is modest, it can be practical to identify each of them, and to ameliorate human fallibility through peer review.

- If the number of infeasible test obligations is large, it becomes impractical to carefully reason about each to avoid excusing an obligation that is feasible but difficult to satisfy. A common practice is to measure the extent to which a test suite approaches an adequacy criterion.
- Quantitative measures of test coverage are widely used in industry. They are simple and cheap to calculate

COMPARING CRITERIA

- It would be useful to know whether one test adequacy criterion was more effective than another in helping find program faults,two kinds of answers to such a question, empirical and analytical.
- An empirical answer would be based on extensive studies of the effectiveness of different approaches to testing in industrial practice, it doesn't give clear-cut answers
- An analytical answer to questions of relative effectiveness would describe conditions under which one adequacy criterion is guaranteed to be more effective than another, or describe in statistical terms their relative effectiveness.
- Analytic comparisons of the strength of test coverage depends on a precise definition of what it means for one criterion to be “stronger” or “more effective” than another
- Let us first consider single test suites. In the absence of specific information, we cannot exclude the possibility that any test case can reveal a failure. A test suite TA that does not include all the test cases of another test suite TB may fail revealing the potential failure exposed by the test cases that are in TB but not in TA.
- Many different test suites might satisfy the same coverage criterion. To compare criteria, then, we consider all the possible ways of satisfying the criteria.
- If every test suite that satisfies some criterion A is a super-set of some test suite that stratifies criterion B, , then we can say that A “subsumes” B.
- Test coverage criterion A subsumes test coverage criterion B iff, for every program P, every test set satisfying A with respect to P also satisfies B with respect to P.