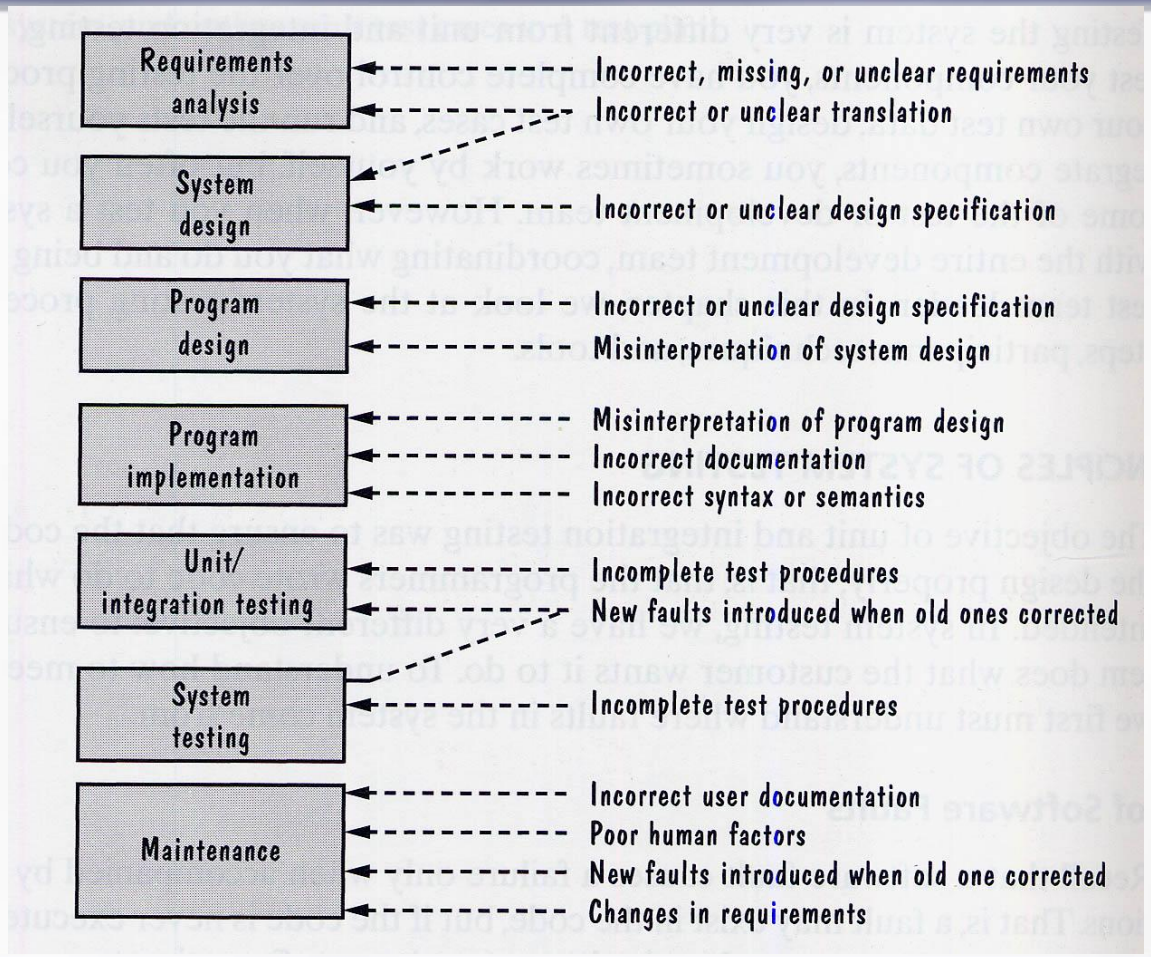
System Testing

Unit & Integration Testing

Objective: to make sure that the program code implements the design correctly.

System Testing Objective: to ensure that the software does what the customer wants it to do.

Causes of Software Faults



System Testing Process

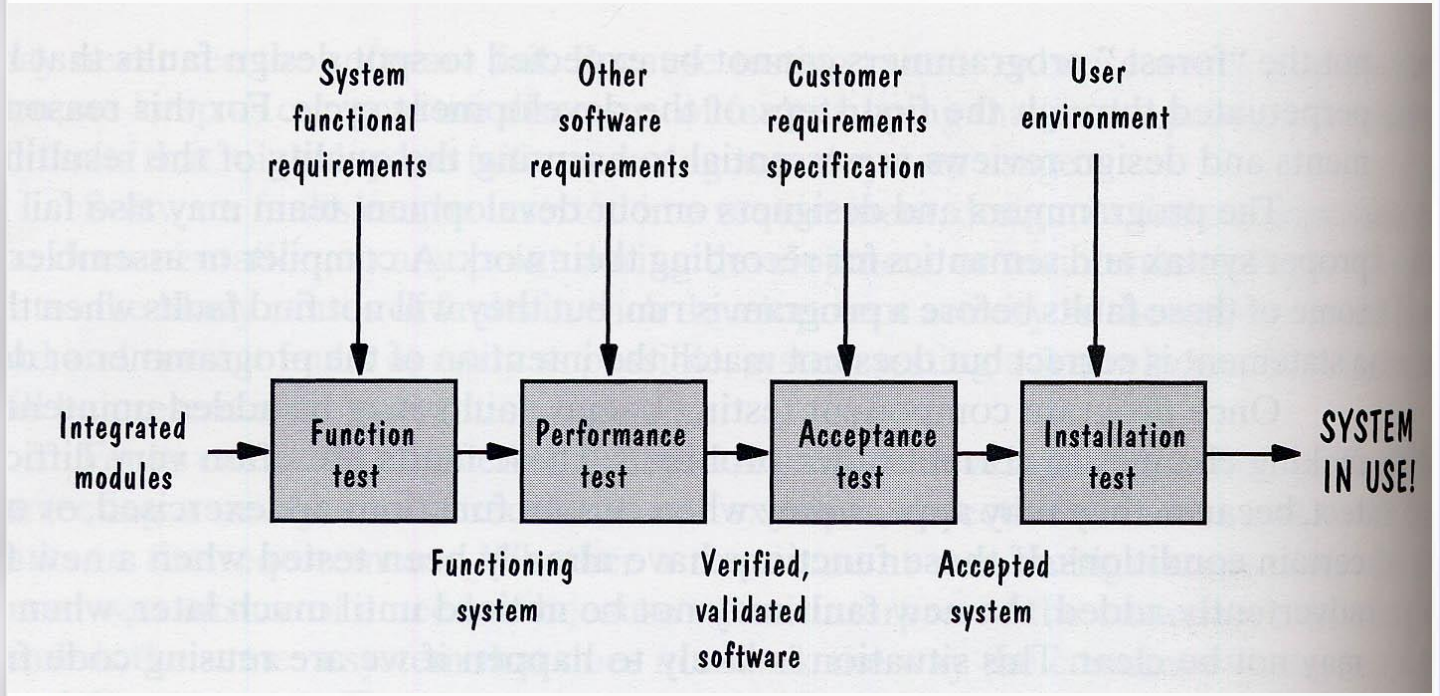
Function Testing: the system must perform functions specified in the requirements.

Performance Testing: the system must satisfy security, precision, load and speed constrains specified in the requirements.

Acceptance Testing: customers try the system (in the lab) to make sure that the system built is the system they requested.

Deployment Testing: the software is deployed and tested in the production environment.

System Testing Process



Build & Integration Plan

- Large complex system is very hard to test.

+ We can devise a build & integration plan defining subsystems to be built and tested such that we are managing a less complex entity. Such subsystems usually correspond to increments in system scope & functionality and are called spins.

Example: Microsoft's Synch & Stabilize approach.

Build & Integration Plan Cont'd

- Each spin is identified including its functionality and testing schedule.
- If the test of spin *n* succeeds but the problem arises during the testing of spin *n+1* we know that the fault comes from components / functionality introduced in spin *n+1*.
- Accounting System Example: spin 0 – basic accounting operations are tested; spin 1 – accounting with data auditing tested; spin 2 – access user control is tested, etc.

Configuration Management

Module-based systems can be shipped in various configurations of components defining the system's functionality and performance.

Different configurations are referred to as *product* versions or *product editions*.

Sequential modifications of a configuration corresponding to product releases are identified as *major* versions.

Sequential modifications of the same major release configuration are identified by *minor* versions.

During development & testing current pre-release version is identified by the build number.

Product Edition **V** *Major* **.** *Minor* **.** *Build Number*

E.g. Sonar Producer v 5.1.1012

Keeping track of version is a must for phased development!

Regression Testing

Regression testing ensures that no new faults are introduced when the system is modified, expanded or enhanced as well as when the previously uncovered faults are corrected.

Procedure:

1. Insert / modify code
2. Test the directly affected functions
3. Execute all available tests to make sure that nothing else got broken

Invaluable for phased development!

Unit testing coupled with nightly builds serves as a regression testing tool for agile methodology.

Source & Version Control

Open Source: CVS flavors

Microsoft: Visual Source Safe, TFS

- Each version is stored separately
- Delta changes
- Modification comments
- Revision history including users

Do not use conditional compilation for tasks other than debugging, tracing or checked builds.

Test Team

Testers: devise test plans, design, organize and run tests

Analysts: assist testers, provide guidance for the *process* verification as well as on the appearance of the test results

Designers: assist testers, provide guidance for testing components and software architecture

Users: provide feedback for the development team

Function Testing

Thread Testing: Verifying a narrow set of actions associated with a particular function

Test process requirements

- Test plan including the stopping criteria
- High probability for detecting a fault
- Use of independent testers
- The expected actions and the expected output must be specified
- Both valid and invalid input must be tested
- System must not be modified in the testing process (or to make testing easier)

Performance Testing

- **Load / Stress Testing:** large amount of users / requests
- **Volume Testing:** large quantities of data
- **Security Testing:** test access, authentication, data safety
- **Timing Testing:** for control and event-driven systems
- **Quality / Precision Testing:** verify the result accuracy (when applies)
- **Recovery Testing:** verify recovery from failure process (when applies)

Reliability,Availability,Maintainability

Reliability: the probability of software operating without failure under given conditions *for* a given time interval *T*.

R = MTTF/(T+MTTF)

Availability: the probability of software operating without failure *at* a given moment in time.

MT(Between Failures) = MT(To Failure)+MT(To Repair)

A = MTBF/(T+MTTF)

Maintainability: the probability of successful software maintenance operation being performed within a stated interval of time.

M = T/(T+MTTR)

Acceptance Testing

The purpose is to enable customers and users to determine if the system built really meets their needs and expectations.

Benchmarking: a predetermined set of test cases corresponding to typical usage conditions is executed against the system

Pilot Testing: users employ the software as a small-scale experiment or in a controlled environment

Alpha-Testing: pre-release closed / in-house user testing

Beta-Testing: pre-release public user testing

Parallel Testing: old and new software are used together and the old software is gradually phased out.

Acceptance testing uncovers requirement discrepancies as well as helps users to find out what they really want (hopefully not at the developer's expense!)

Deployment Testing

The software is installed on a target system and tested with:

- Various hardware configurations
- Various supported OS versions and service packs
- Various versions of third-party components (e.g. database servers, web servers, etc.)
- Various configurations of resident software

Safety-Critical Systems

Safe means free from accident or loss.

Hazard: a system state that, together with the right conditions, can lead to an accident.

Failure Mode: a situation that can lead to a hazard.

We can build a fault tree to trace known failure modes to unknown effects / hazards.

Safety-Critical System Issues

Remember Murphy's Laws:

- If a fault can occur it will
- If a user can make a mistake he will
- Least probable fault causes are not

* 100% reliable system is not necessarily safe or secure!
* Budget with testing in mind.

Hazard and Operability Studies

HAZOPS involves structured analysis to anticipate system hazards and to suggest means to avoid or deal with them.

During testing we must select test cases to exercise each failure mode to make sure that the system reacts in a safe manner.

Cleanroom

Method devised by IBM for developing high-quality software with a high-productivity team.

Principles:

1) Software must be *certified* with respect to the specifications (rather than tested)
2) Software must be zero-fault.

Cleanroom Process

1) The software as specified as a black box, then refined as a state box, then refined as a clear box.
2) The box structure encourages analysts to find flaws and omissions early in the project life cycle (when it is cheaper and faster to fix them).
3) The clear-box specification is converted into an intended function using natural language or mathematic notation.
4) For each function a correctness theorem is devised and proven.

* Unit testing is not necessary or even permitted!
* Errors found by statistical testing tends to be simple mistakes that are easy to fix as the cleanroom eliminates deep principle problems.

IBM claims an order of magnitude fault reduction (e.g. 3 faults per 1000 lines of code).