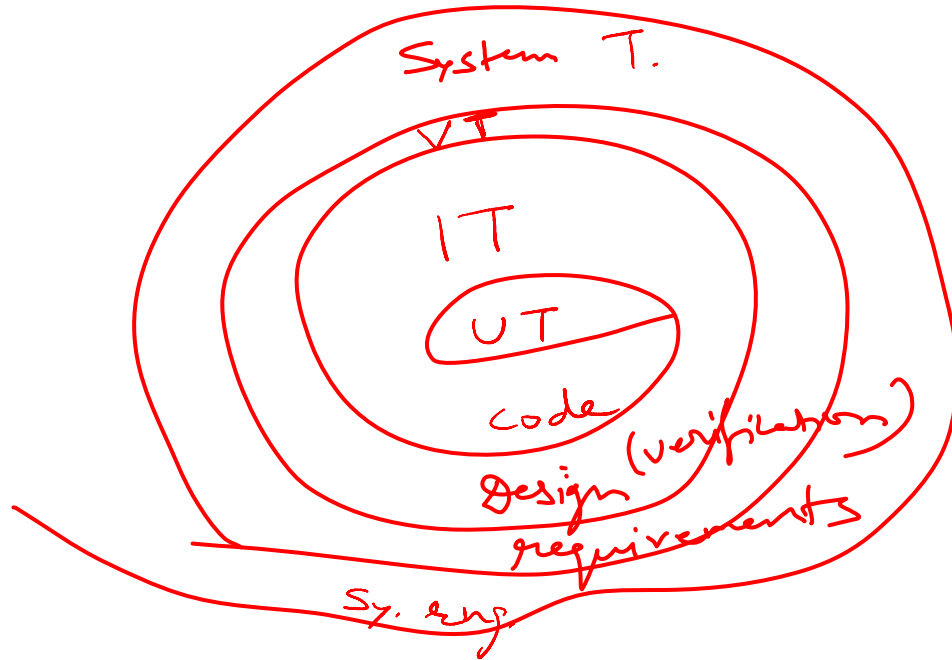


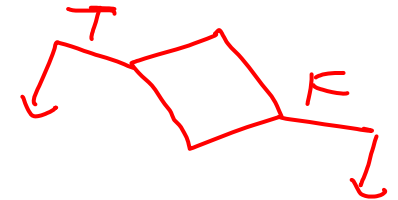
# **Software Testing & Risk Analysis**

# Testing Strategy



unit (single module)  
Integr. (combined " )

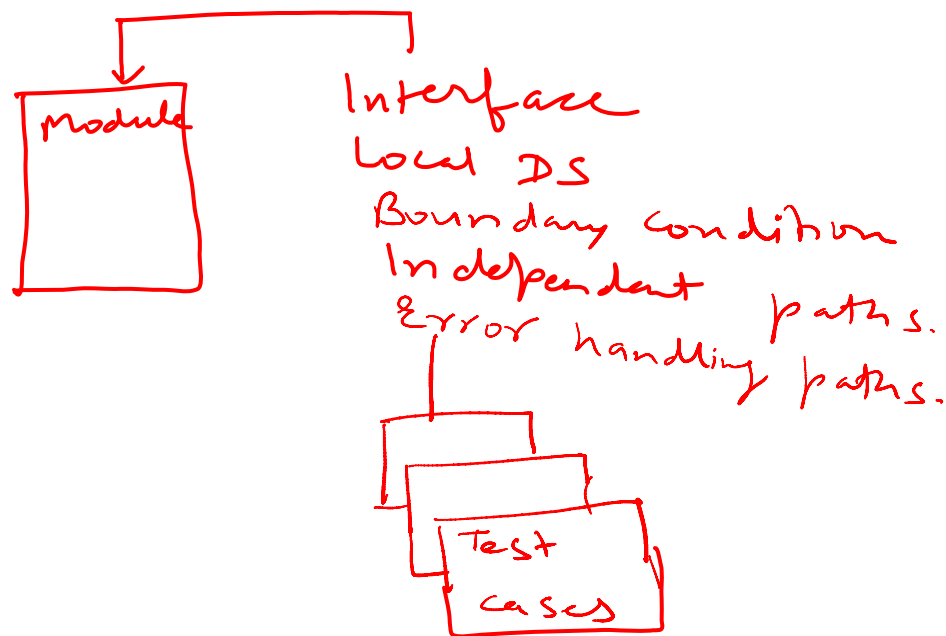
I/p & requ. Black box  
Internal logic White box



Validation → requirements.  
Verification → functions

# Unit testing

Unit testing is undertaken after a module has been coded and successfully reviewed. Unit testing (or **module testing**) is the testing of different units (or modules) of a system in isolation.



# Levels of Testing

- Performance testing
- Stress testing
- Volume testing
- Configuration testing
- Compatibility testing
- Regression testing
- Recovery testing
- Maintenance testing
- Documentation testing
- Usability testing

# Performance testing

- Performance testing is carried out to check whether the system needs the **nonfunctional requirements** identified in the SRS document. All performance tests can be considered as black-box tests.

# Stress Testing

Stress testing is also known as endurance (toleration) testing. Stress testing evaluates system performance when it is **stressed for short periods of time**. Stress tests are **black box tests** which are designed to impose a range of **abnormal and even illegal input conditions** so as to stress the capabilities of the software. Input data volume, input data rate, processing time, utilization of memory, etc. are tested beyond the designed capacity.

For example, if the non-functional requirement specification states that the response time should not be more than **20 secs per transaction when 60 concurrent users are working**, then during the stress testing the response time is checked with 60 users working simultaneously.

For example, suppose an **operating system is supposed to support 15 multiprogrammed jobs**, the system is stressed by attempting to run 15 or more jobs simultaneously. A real-time system might be tested to determine the effect of simultaneous arrival of several high-priority interrupts.

# Volume Testing

It is especially important to check whether the **data structures** (arrays, queues, stacks, etc.) have been designed to **successfully extraordinary situations**.

For example, a **compiler** might be tested to check whether the **symbol table overflows** when a very large program is compiled.

# Configuration Testing

This is used to analyze system behavior in various **hardware and software configurations specified** in the requirements. Sometimes systems are built in variable configurations for different users. For instance, we might define a minimal system to serve a single user, and other extension configurations to serve additional users. The system is configured in each of the required configurations and it is checked if the system behaves correctly in all required configurations.

# Compatibility Testing

This type of testing is required when the **system interfaces with other types of systems**. Compatibility aims to check whether the interface functions perform as required. For instance, if the **system needs to communicate with a large database system** to retrieve information, compatibility testing is required to test the **speed and accuracy** of data retrieval.

# Regression Testing

This type of testing is required when the system being tested is an **upgradation of an already existing system** to fix some bugs or enhance functionality, performance, etc. Regression testing is the practice of **running an old test suite after each change to the system or after each bug fix to ensure that no new bug has been introduced due to the change or the bug fix.** However, if only a few statements are changed, then the entire test suite need not be run - only those test cases that test the functions that are likely to be affected by the change need to be run.

# Recovery Testing

Recovery testing tests the **response of the system to the presence of faults, or loss of power, devices, services, data, etc.** The system is subjected to the loss of the mentioned resources (as applicable and discussed in the SRS document) and it is checked if the system recovers satisfactorily.

For example, the **printer can be disconnected** to check if the system hangs. Or, the **power may be shut down** to check the extent of data loss and corruption.

## Maintenance Testing

This testing addresses the diagnostic programs, and other procedures that are required to be developed to help maintenance of the system. It is verified that the artifacts exist and they perform properly.

## Documentation Testing

It is checked that the required **user manual, maintenance manuals, and technical manuals exist and are consistent**. If the requirements specify the types of audience for which a specific manual should be designed, then the manual is checked for compliance.

# Usability Testing

Usability testing concerns checking the user interface to see if it meets **all user requirements** concerning the **user interface**. During usability testing, the display screens, report formats, and other aspects relating to the user interface requirements are tested.

## Characteristics of a good software engineer

The attributes that good software engineers should possess are as follows:

- Exposure to systematic techniques, i.e. familiarity with software engineering principles.
- Good technical knowledge of the project areas (Domain knowledge).
- Good programming abilities.
- Good communication skills: These skills comprise of oral, written, and interpersonal skills.
- High motivation.
- Sound knowledge of fundamentals of computer science.
- Intelligence.
- Ability to work in a team.
- Discipline, etc.

# Risk management

- **Project risks:** Project risks concern various forms of **budgetary, schedule, personnel, resource, and customer-related problems**. An important project risk is schedule slippage. Since, software is intangible, it is very difficult to monitor and control a software project. It is very difficult to control something which cannot be seen.
- **Technical risks:** Technical risks concern **potential design, implementation, interfacing, testing, and maintenance problems**. Technical risks also include ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence (outdated). Most technical risks occur due to the development team's insufficient knowledge about the project.
- **Business risks:** This type of risks include risks of building an **excellent product, losing budgetary or personnel commitments, etc.**

# Risk assessment(estimation)

- The objective of risk assessment is to **rank the risks** in terms of their damage causing potential. For risk assessment, first each risk should be rated in two ways:
  - The likelihood of a **risk coming true**
  - The consequence of the **problems associated with that risk**

# Risk containment(prevention)

- After all the identified risks of a project are assessed, plans must be made to contain the **most damaging and the most likely risks**. Different risks require different containment procedures.
- There are three main strategies to plan for risk containment:
  - **Avoid the risk:** This may take several forms such as discussing with the customer to **change the requirements** to reduce the scope of the work, giving incentives to the engineers to avoid the risk of manpower turnover, etc.
  - **Transfer the risk:** This strategy involves getting the risky component developed by a **third party**, buying insurance cover, etc.
  - **Risk reduction:** This involves planning ways to contain the damage due to a risk. For example, if there is risk that some key **personnel might leave**, new recruitment may be planned.

# SRS

## Software Requirements Specification for

**<Project>**

**Prepared by <author>**

**<organization>**

**<date created>**

<b>Table of Contents</b>	<b>ii</b>
<b>Revision History</b>	<b>iii</b>
<b>1. Introduction</b>	<b>1</b>
1.1 Purpose	1
1.2 Document Conventions	1
1.3 Intended Audience and Reading Suggestions	1
1.4 Product Scope	1
1.5 References	1
<b>2. Overall Description</b>	<b>2</b>
2.1 Product Perspective	2
2.2 Product Functions	2
2.3 User Classes and Characteristics	2
2.4 Operating Environment	2
2.5 Design and Implementation Constraints	2
2.6 User Documentation	2
2.7 Assumptions and Dependencies	3

<b>3. External Interface Requirements</b>	<b>3</b>
3.1 User Interfaces	3
3.2 Hardware Interfaces	3
3.3 Software Interfaces	3
3.4 Communications Interfaces	3
<b>4. Domain Model</b>	<b>4</b>
<b>5. System Features (Use Cases)</b>	<b>4</b>
5.1 Use Case 1	4
5.2 Use Case 2 (and so on)	5
<b>6. Other Nonfunctional Requirements</b>	<b>5</b>
6.1 Performance Requirements	5
6.2 Safety Requirements	5
6.3 Security Requirements	5
6.4 Software Quality Attributes	5
<b>7. Other Requirements</b>	<b>5</b>
<b>Appendix A: Glossary</b>	<b>6</b>
<b>Appendix B: Analysis Models</b>	<b>6</b>
<b>Appendix C: To Be Determined List</b>	<b>6</b>

- 1.1 Identify the product whose software requirements are specified in this document
- 1.2 For example, state whether priorities for higher-level requirements are assumed to be inherited by detailed requirements, or whether every requirement statement is to have its own priority
- 1.3 Describe the different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers
- 1.4 The different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers
- 1.5 List any other documents or Web addresses to which this SRS refers

- 2.1 Describe the context and origin of the product being specified in this SRS.  
A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful
- 2.2 Summarize the major functions the product must perform or must let the user perform
- 2.3 Identify the various user classes that you anticipate will use this product.  
User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience.
- 2.4 Describe the environment in which the software will operate, including the hardware platform, operating system and versions
- 2.5 Describe any items or issues that will limit the options available to the developers. These might include: hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards. for example, if the customer's organization will be responsible for maintaining the delivered software.
- 2.6 List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software

- 2.6 These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project.
- 3.1 Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on.
- 3.2 Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system.
- 3.3 Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components
- 3.4 Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on.

6.1 If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices

You may need to state performance requirements for individual functional requirements or features

6.2 Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Define any safety certifications that must be satisfied

6.3 Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any security or privacy certifications that must be satisfied

6.4 Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability

- **Other Requirements**

Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.

- **Appendix A: Glossary**

Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.

- **Appendix B: Analysis Models**

Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.

- **Appendix C: To Be Determined List**

Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.