



Software Engineering

Rajiv Mulay

What is Software?

Software is:

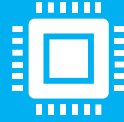
- Instructions (computer programs) when executed, provide desired function and performance.
- Data structures that enable the programs to adequately manipulate information, and
- Documents that describe the operation and use of programs.

Software Characteristics

- Software is developed or engineered
- Software doesn't "wear out".
- Most software is custom-built



Software Engineering



The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software. *IEEE*

Standards Collection: Software Engineering, IEEE Standard 610.12-1990, IEEE, 1993).



Ensure Quality, Timely delivery,
Within budget

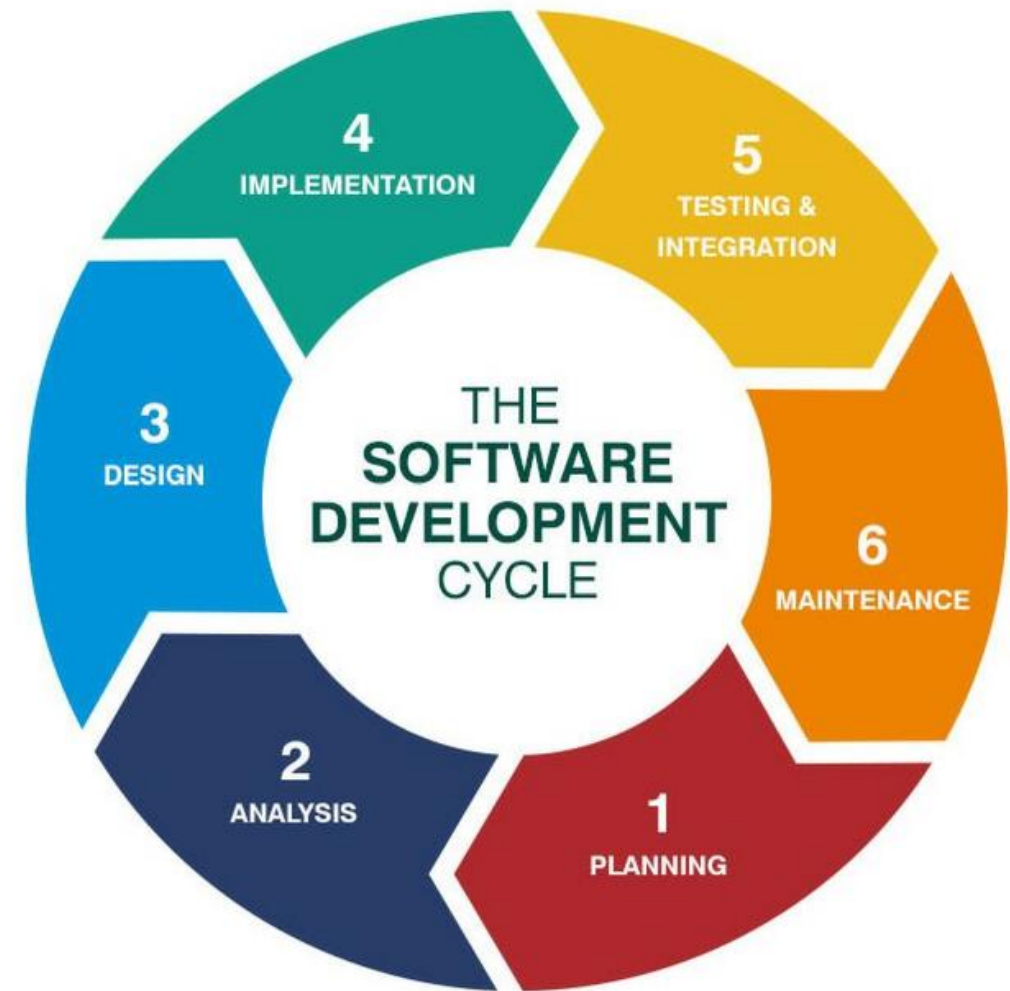


Employ a set of processes,
methods and tools.



Software Development Lifecycle

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software's.



Software Development Stages

Planning

Requirements

Design

Implementation

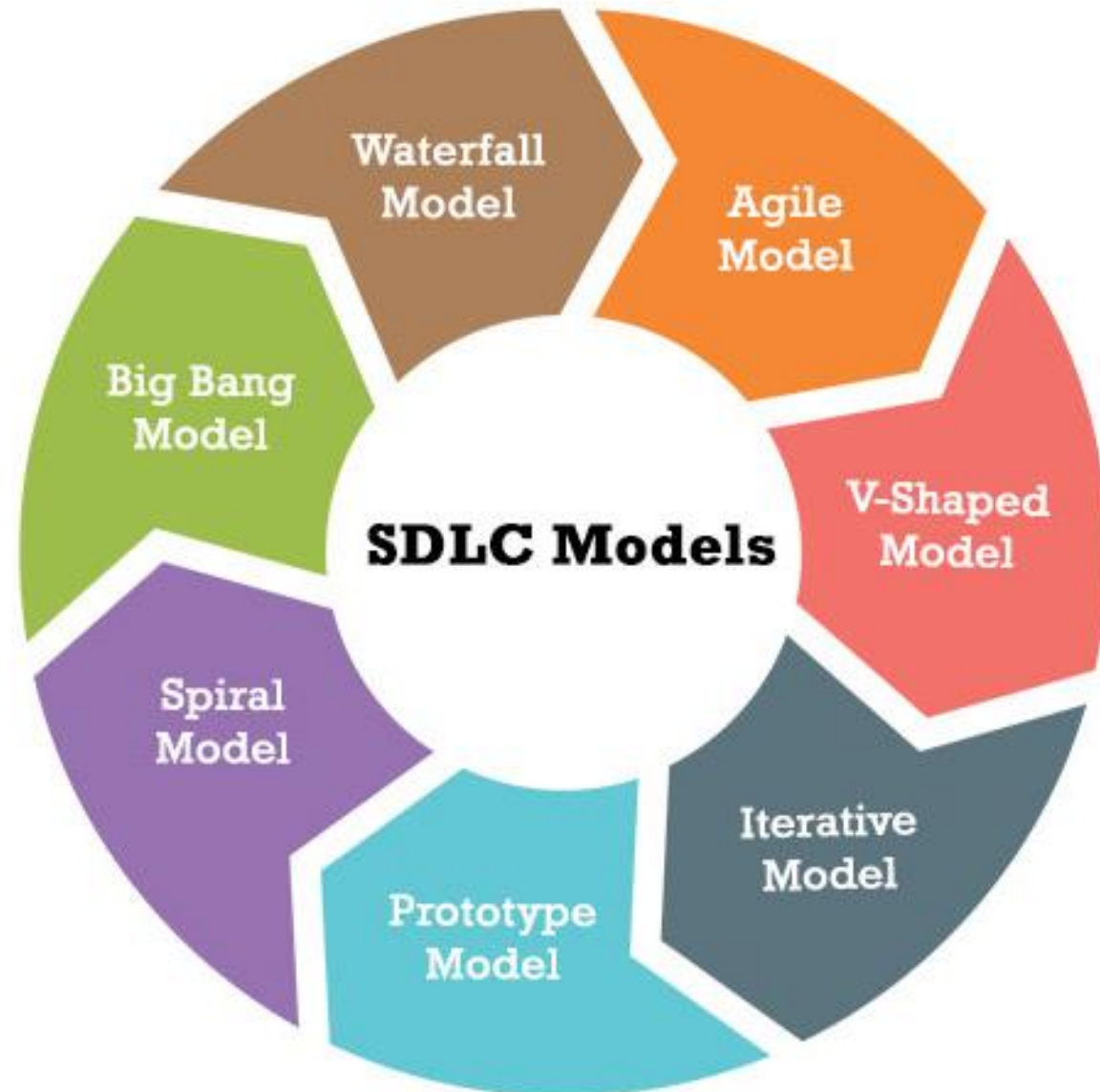
Testing

Release &
Maintenance

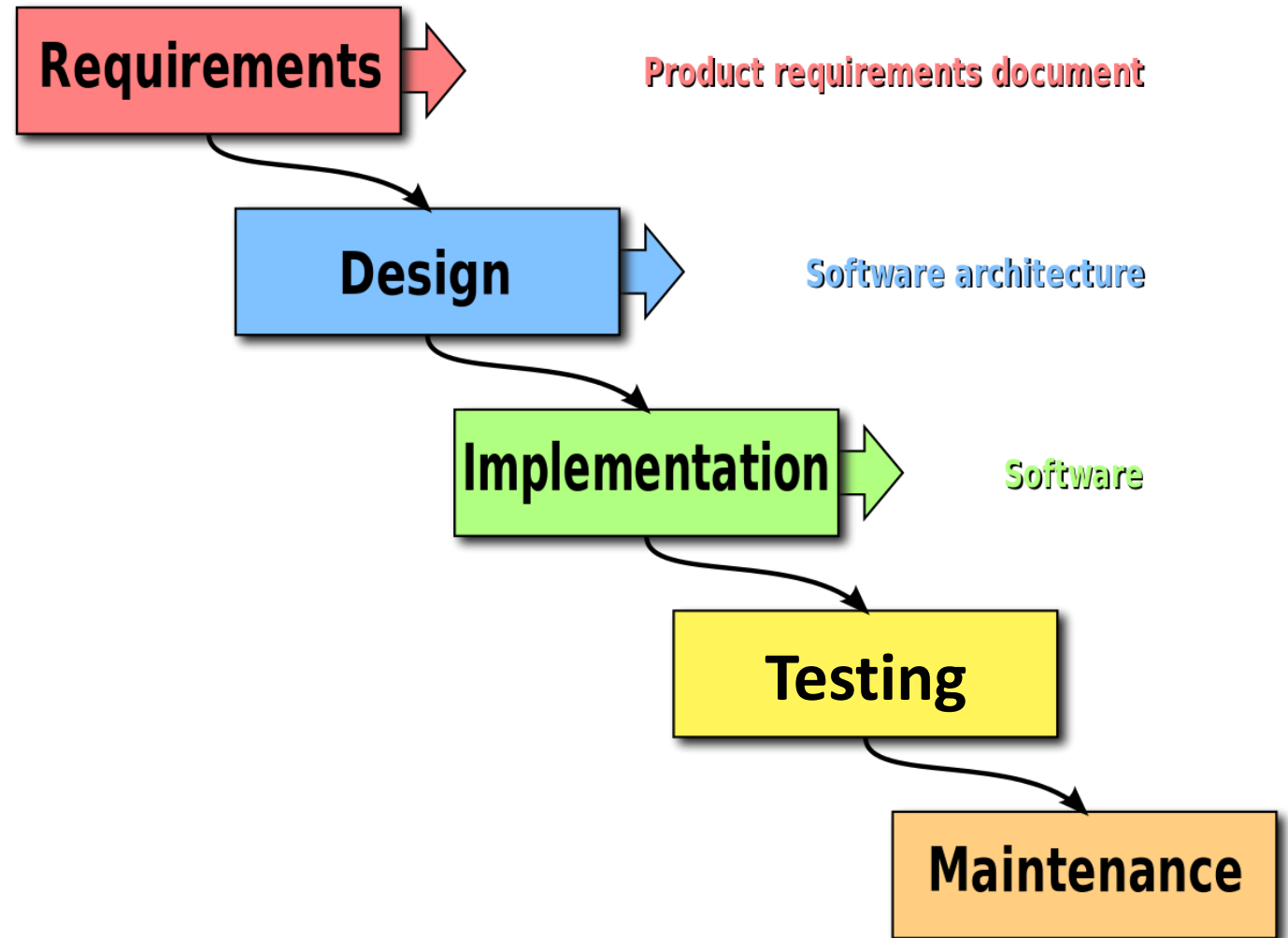


Software Models

SDLC Models



Water Fall Model



Use Cases for Water Fall Model



**THE REQUIREMENTS
ARE PRECISELY
DOCUMENTED**



**PRODUCT
DEFINITION IS
STABLE**



**THE TECHNOLOGIES
STACK IS
PREDEFINED**



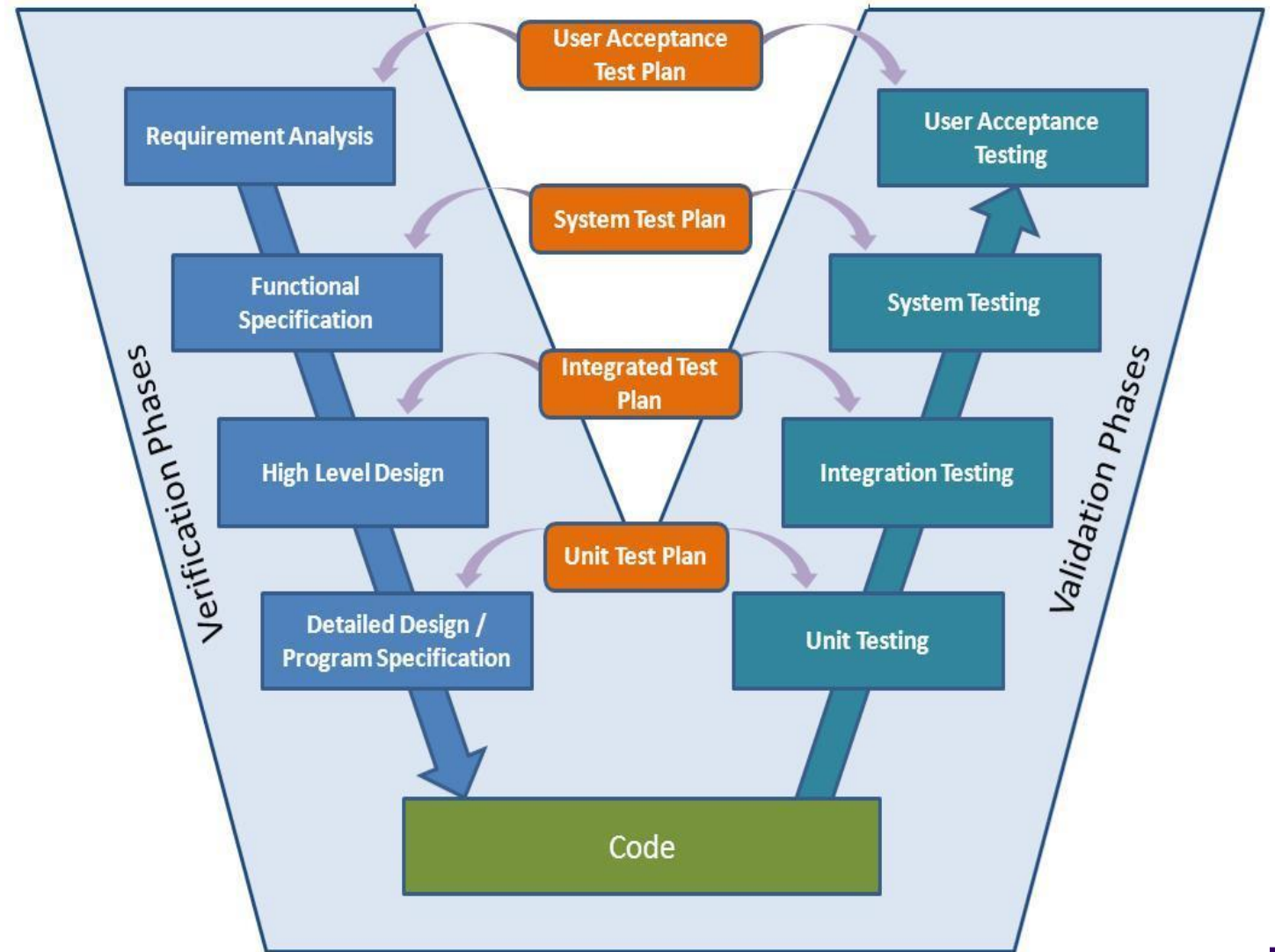
**NO AMBIGUOUS
REQUIREMENTS**



**THE PROJECT IS
SHORT**



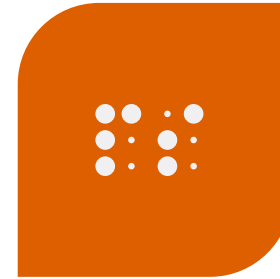
V Model



Use Cases for V-Model



**FOR THE PROJECTS WHERE
AN ACCURATE PRODUCT
TESTING IS REQUIRED**



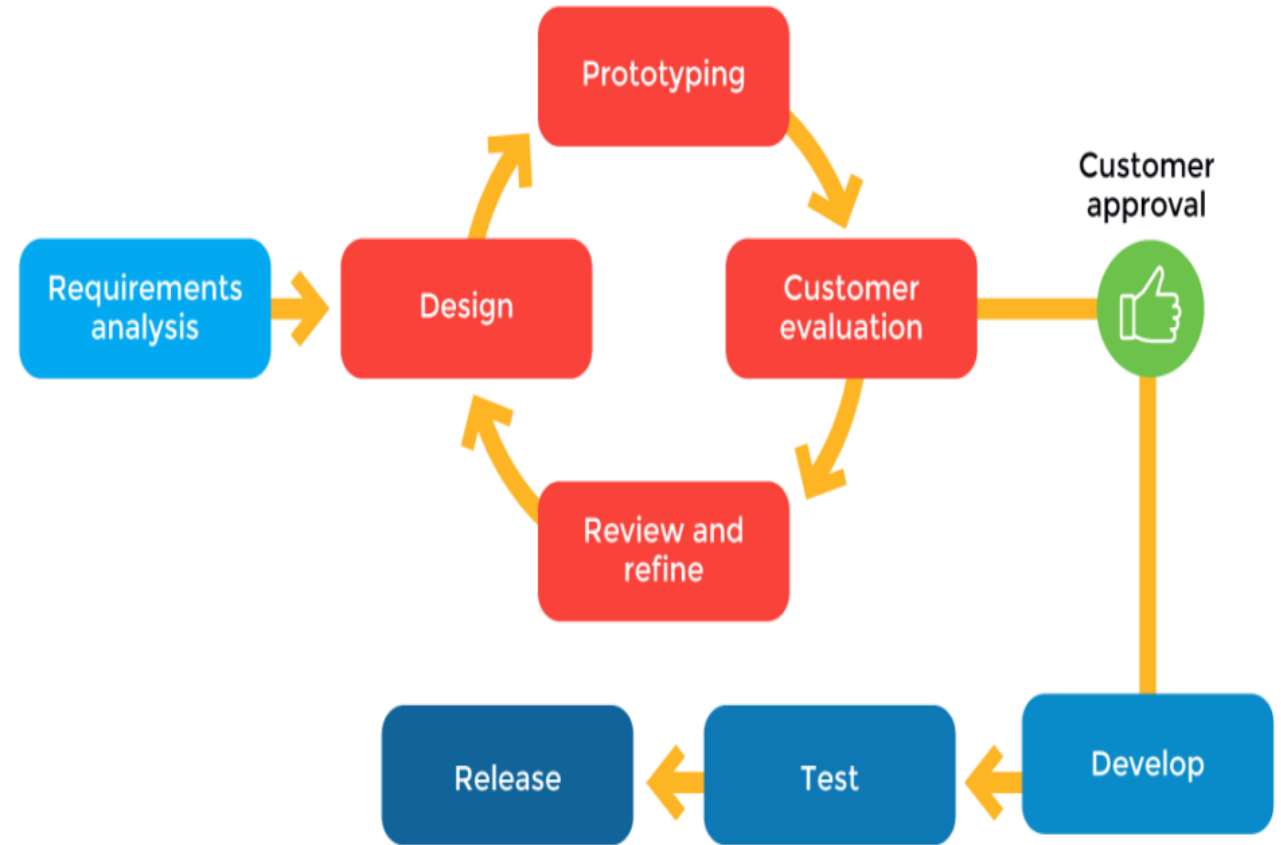
**REQUIREMENTS ARE
STRICTLY PREDEFINED**



**FOR THE SMALL AND MID-
SIZED PROJECTS**



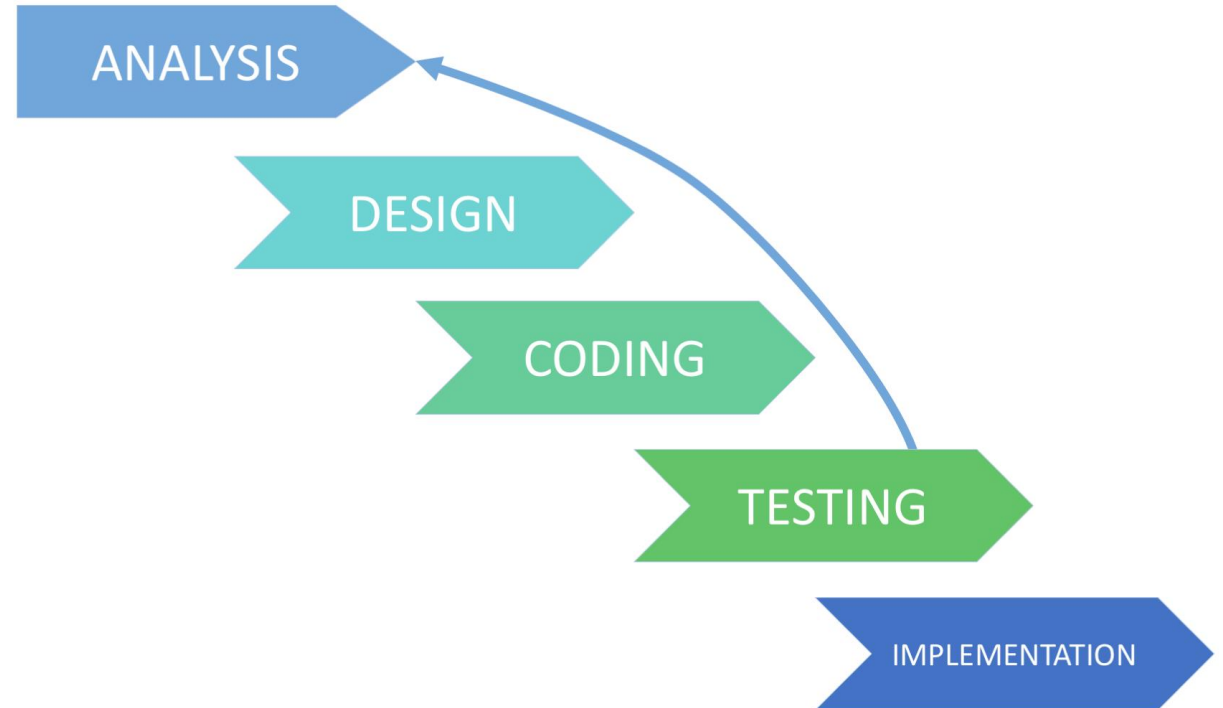
Prototype Model



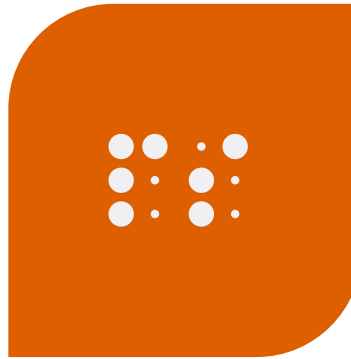
- Intermediate working models are delivered and refined to build the final functional system
- Incomplete versions of the application being developed are built for soliciting early feedback



Iterative Models



Use Cases for Iterative Models



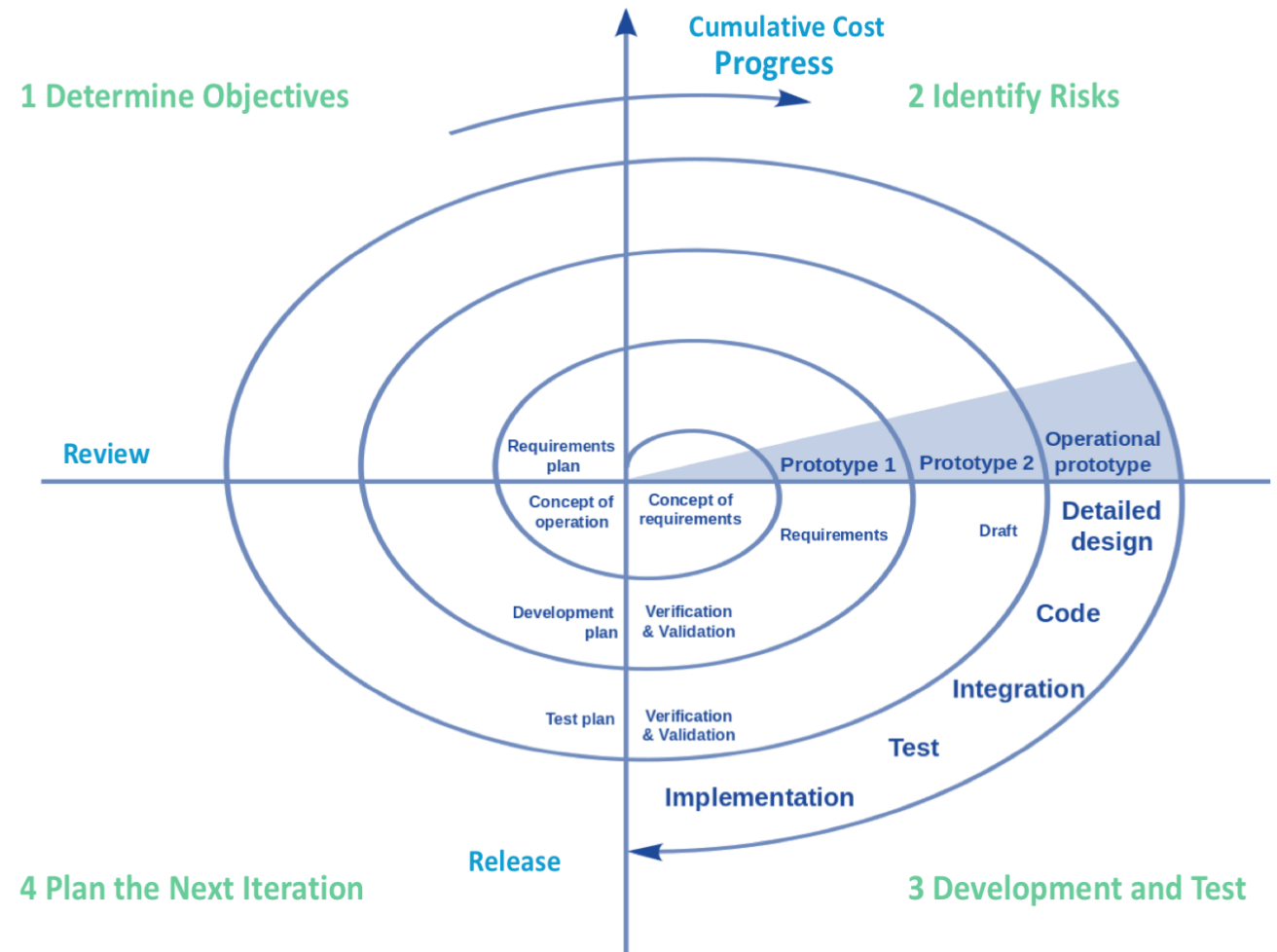
**APPLIED TO THE LARGE-SCALE
PROJECTS**



**THE MAIN TASK IS PREDEFINED, BUT
THE DETAILS MAY ADVANCE WITH
TIME**

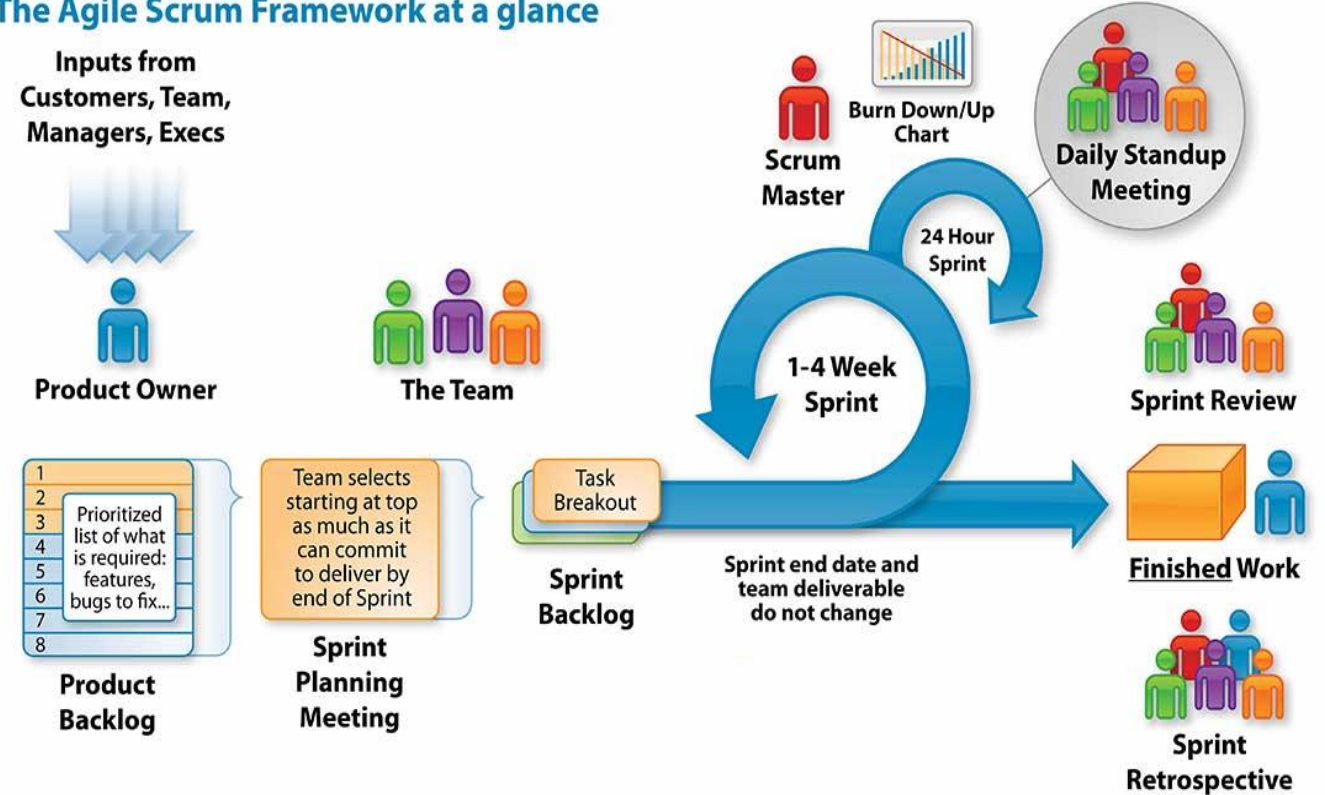


Spiral Model



Agile Scrum

The Agile Scrum Framework at a glance





Requirement Management

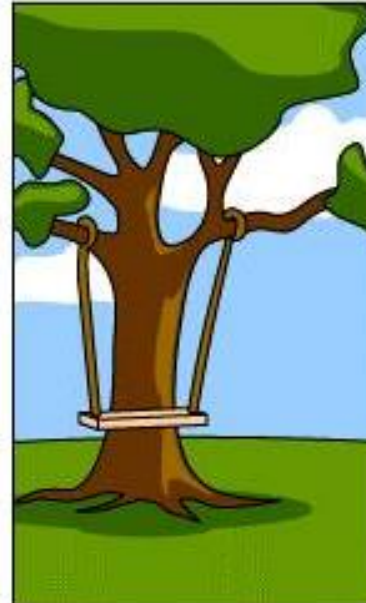
Requirement (Mis) Management



What the customer really needed



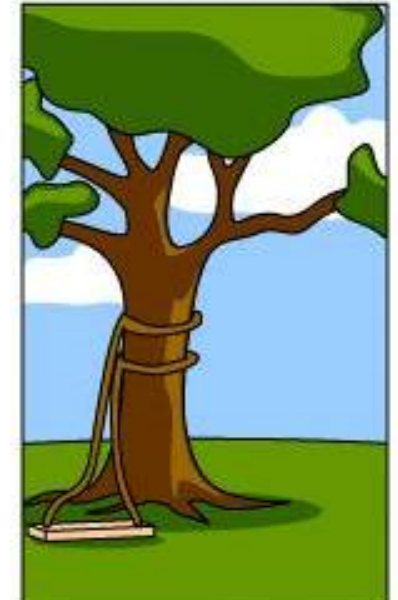
How the customer explained it



How the Project Leader understood it



How the Analyst designed it



How the Programmer wrote it



What is Requirement?



A requirement is defined as a property that must be exhibited in order to solve some real-world problem.



S/W requirements express the needs and constraints placed on a SW product that contribute to the solution of some real-world problem



A requirement is a singular documented physical and functional need that a design, product or process must be able to perform.



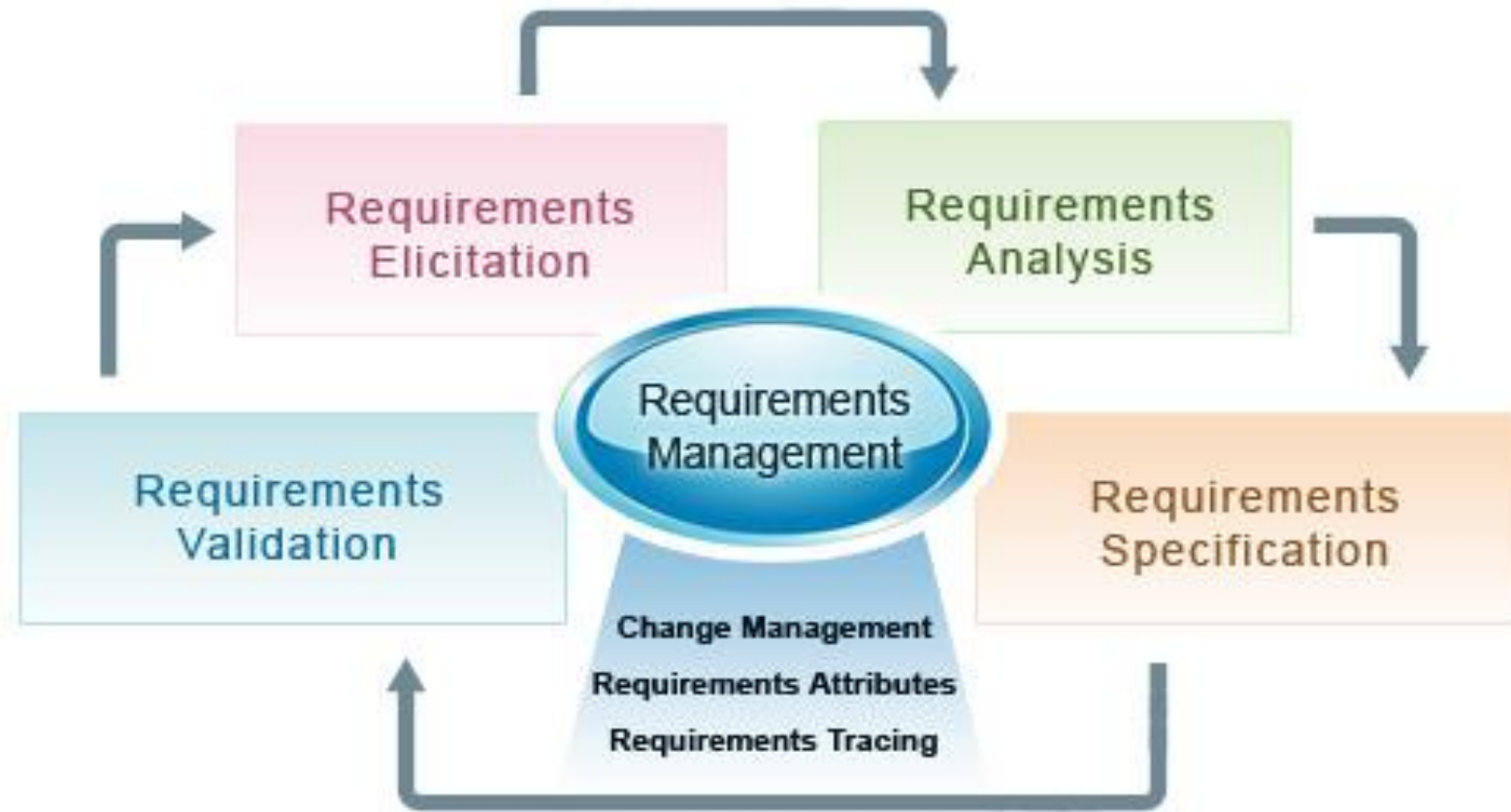
A Requirements specification is the direct result of a requirements analysis.



A Software requirements specification (SRS) is a complete description of the behavior of a system to be developed and may include a set of use cases and non-functional requirements.



Requirement Management



Types of Requirement

Functional

- Functionality/Feature Supported
- Implicit Requirements

Inverse

- Negative requirements
- Clarity of scope

Non-Functional

- Performance, Security
- Reliability, Maintainability
- Availability, Accuracy

Constraints

- Memory Constraints
- Power Constraints



Requirement Traceability

- The requirements should be **Tagged**
 - Design to the Requirement
 - Code should trace to design
 - Unit test cases to code
 - System test cases to the Requirements
-
- Questions that can be answered with the help of traceability info:
 - What is the impact of a requirement change?
 - Are all requirements allocated?
 - What acceptance test will be used to test a requirement?
 - Is this design element necessary?



*To Ensure
Complete
Test
Coverage*



A project without adequately traced work products is difficult to sustain



Requirement Change Management

- All requirements are base-lined
- Every requirement change should follow the cycle:
 - Proposal for change
 - Impact analysis (effort, schedule, cost)
 - Go / no go decision by change control board
 - Required changes to plan, estimates
 - Required change to design / test cases
 - Required change traceability / other documents
 - Software changes
 - Other steps as per SDLC





Design

What is Software Architecture

Software architecture refers to the fundamental structures of a **software** system and the discipline of creating such structures and systems. Each **structure** comprises **software** elements, relations among them, and properties of both elements and relations.

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other.



What is Software Design

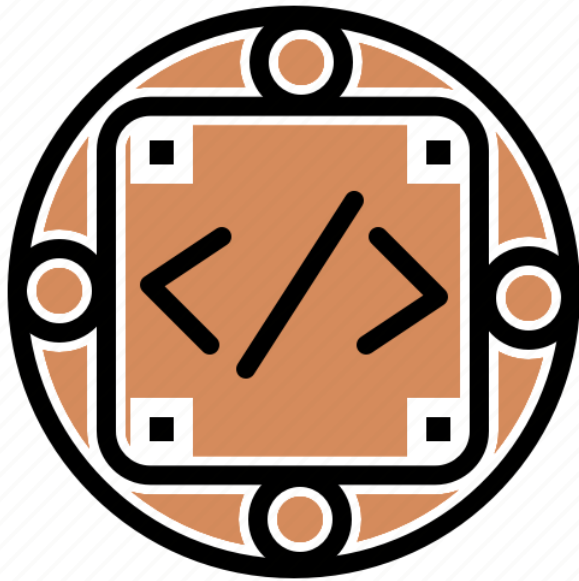
Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

Modularize - is a technique to divide a software system into multiple discrete and independent modules
Concurrency – Modules executing in parallel
Cohesion - is a measure that defines the degree of intra-dependability within elements of a module.
Coupling - is a measure that defines the level of inter-dependability among modules of a program



Implementation

Key Objective of Implementation



- Translate the design into code that will satisfy requirements. The software implemented should be
 - **Complete:** everything that is in the requirements and design is implemented
 - **Consistent:** no mismatched interfaces and consistent with the design
 - **Stylistic:** exhibits good programming style (e.g., safe constructs, information hiding, well defined types, reasonable module sizes and complexity, well-defined module interfaces, avoids side effects)
 - **Understandable:** code should be constructed that is easy to read, not necessarily easy to write
 - **Modifiable:** again this is a living document
 - **Confirmable, Verifiable and testable:** you can tell when you've met the design and requirements



Coding Principles and Concepts

Before you write one line of code be sure you -

Understand the problem you're trying to solve

Avoid developing an elegant program that solves the wrong problem.

Understand basic design principles and concepts

Create a set of unit tests that will be applied once the component you code is completed

Test cases can be automated or manual.

As you begin writing code, remember

Code is read much more often than it is written

The goal of coding guidelines is to improve the productivity of all software development: Easier, more reliable, faster

Comment as you go. It only takes a few seconds. Don't wait for later

Source code is a language for people, not just computers

- Ask yourself: "How will the next person know that?"



Coding Principles and Concepts

As you begin writing code, be sure you,

- Constrain your algorithms by following structured programming practice
- Select data structures that will meet the needs of the design
- Keep conditional logic as simple as possible
- Create nested loops in a way that makes them easily testable
- Select meaningful variable names and follow other local coding standards
- Write code that is self-documenting out
- Create a visual layout (eg. indentation and blank spaces) that aids understanding

Validation principles: After you've completed your coding pass, be sure you,

- Conduct a code walk through when appropriate
- Perform unit tests and correct errors you've uncovered
- Look for correctness
- Ensure coding standards have been maintained
- Ensure that the code is self-documenting
- Refactor the code



Unit Testing

UNIT TESTING FOCUSES VERIFICATION EFFORT ON THE SMALLEST UNIT OF SOFTWARE DESIGN – THE SOFTWARE COMPONENT OR DESIGN

USING THE COMPONENT LEVEL-DESIGN AS A GUIDE, IMPORTANT CONTROL PATHS ARE TESTED TO UNCOVER ERRORS WITHIN THE BOUNDARY OF THE MODULE

WHITE-BOX ORIENTED APPROACH



Unit Testing Considerations



Test Module Interfaces

If data do not enter and exit properly, all other tests are moot



Examine Local Data Structures

Local data structures should be exercised and the local impact on global data should be ascertained(if possible) during unit testing



Test Boundary Conditions

Considered to be most important task of unit test step. Test cases that exercise data structure, control flow and data values **just below, at and just above maxima and minima** are very likely to uncover errors



Test all Independent Paths

Selective testing of execution paths is an essential task during the unit test. Test cases should be designed to uncover errors due to erroneous computations, incorrect comparisons, or improper control flow. Basis path and loop testing are effective techniques for uncovering a broad array of path errors



Test all Error Handling Paths

General tendency is to incorporate error handling into software and then 'never test it'. This should not be the case.



Boundary Testing

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

The basic idea in boundary value testing is to select input variable values at their:

Minimum; Just above the minimum; Just below the maximum; Maximum

An exam has a pass boundary at 50 percent, merit at 75 percent and distinction at 85 percent; What would be the boundary conditions



Corner test cases

In engineering, a **corner case** involves a problem or situation that occurs only outside of normal operating parameters

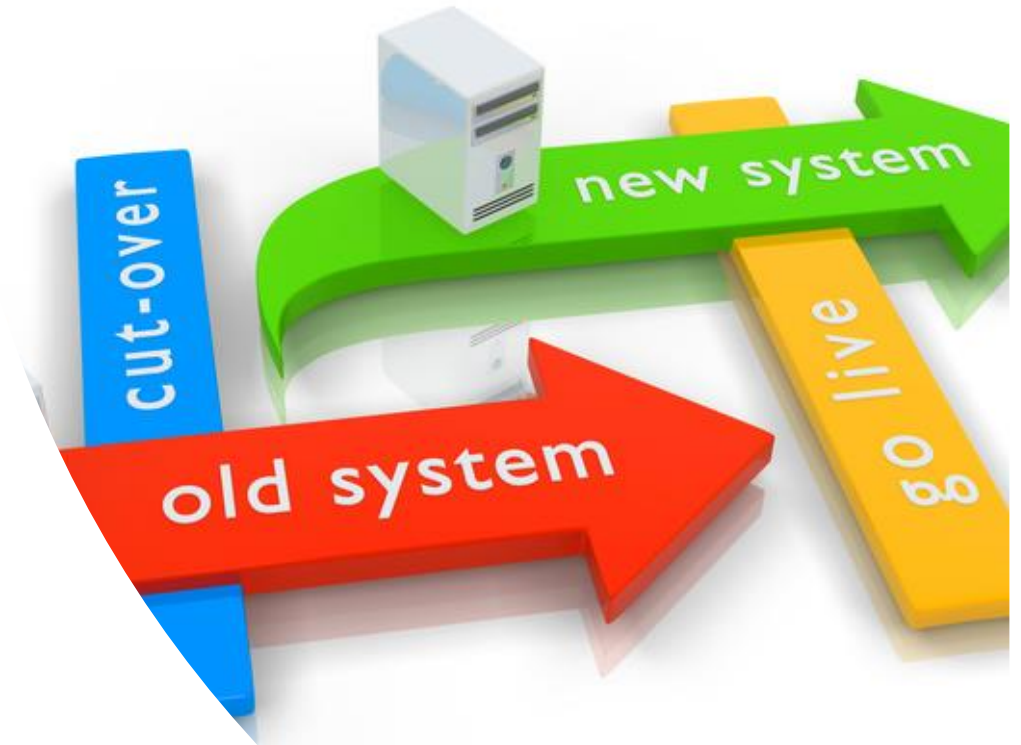
Corner case occurs outside of normal operating parameters



Software Configuration Management

Change – the inevitable part of Software

No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle.



What is Software Configuration Management

The art of coordinating software development to minimize confusion.

Art of identifying, organizing and controlling modifications to the software being built, with the goal of maximizing productivity and minimizing mistakes.

An umbrella activity that forms part of Software Quality Assurance

Consists of Processes, Tools and Techniques to systematically manage the changes to software

Identify all items that collectively define the software configuration and manage changes to these items (*Version Control and Change Control*)



Software Configuration Items

Requirement
Specifications

Project Plan

Design
Specifications

Source Code (and
Executables in
some cases)

Test Specifications,
Plans &
Procedures

Manuals

Supporting
Documents and
Standards



Software Testing

Software Testing

Testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets the required results.



Test Levels – Based on Target of Testing



Unit Testing

- Generally white-box, Stubs are used.
- Done at component level
- Testing of functionality, non functional
- Performed by the development team



Integration Testing

- Testing module interfaces
- Incremental approach
- Non-incremental (big-bang) approach
- Architecture Driven



Test Levels – Based on Target of Testing

System Tests

- Functional System Requirements
- Non-functional system requirements
 - security, speed, accuracy, and reliability.
- Test environment like the production environment
- External interfaces to other applications, utilities, hardware devices, or the operating environment



Test Levels – *Based on Objectives of Testing*



- Acceptance/qualification testing
checks the system behavior against the customer's requirements
- Installation testing
SW verified upon installation in the target environment
Checks HW requirements and installation procedures
- Alpha and beta testing
Given to small, representative set of potential users for trial use
either in-house (*alpha* testing) or external (*beta* testing).
- Regression testing
selective retesting of a system or component to verify that modifications have not caused unintended effects.
- Performance testing
- Stress testing
- Recovery testing
- Usability testing





Release and Maintenance

Release Process

Objectives of a release process

- Provide right version and documentation to customer
- Archive for future use
- Systematic handling of post release defects

Records Created

- Release checklist
- Release notes
- Post release defects



Need of Maintenance

Requirement priorities

Initial systems would not be complete

Flexibility in product/software to allow customer expectations

Changes due to time

Influence of technology, business, competition, late priority realization

Control initial investment

Customer default expectations of post purchase support

Quality expectations



Types of Maintenance

Adaptive

- Modifying the system to cope with changing environment

Perfective

- Implementing new or changed user requirements with functional enhancements

Corrective

- Fixing errors found by users

Preventive

- Increasing maintainability or reliability to prevent problems in future

