LECTURE NOTES

ON

FORMAL METHODS AND SOFTWARE DEVELOPMENT

(CSC 417)

400LEVEL COMPUTER SCIENCE

RHEMA UNIVERSITY (TAKE OFF SITE) ABA

CSC 417- FORMAL METHODS AND SOFTWARE DEVELOPMENT

Outline

1.0 Systematic methods for designing, coding, testing and documenting medium-sized programs

2.0 Major topics include:

i. Formal specification, abstraction, modularity and reusability

Note: students will become strong apprentice programmers able to write a clear specification for a problem, read a specification and design the software to implement it, use appropriate data structures in a program, write reusable code when possible, debug a program and adequately test a program

Overview

Every Software engineering methodology is based on a recommended development process proceeding through several phases such as Analysis, Specification, Design, Coding, Unit Testing, Integration and System Testing, and Maintenance. The use of formal methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analysis can contribute to the reliability and robustness of a design. Formal methods are the use of mathematical modelling for the specification, development and verification of systems in both software and electronic hardware. It is that area of computer science that is concerned with the application of mathematical techniques to the design and implementation of computer hardware and (more usually) software.The formal methods are used to ensure these systems are developed without error.

The mathematical foundation underlying formal methods is used to help ensure the adequacy of the design to result in real world functionality, consistency and dependability in the end product. Much as in other fields of engineering, formal methods applies mathematics to software and hardware engineering in order to add certainty to designing and testing of these systems. Formal methods are used to describe a system's functions prior to design with descriptive languages ensuring the functionality of the system. Formal methods may be used in development depending on the rigor with which the system is described. Formal specifications can function as a guide to requirements. In analysis, formal methods provide the description of functions by which the program can be verified.

The reason for using formal methods

1. **The correctnesss problem**: – producing software that is "correct" is famously difficult and so by using rigorous mathematical techniques, it may be possible to make provably correct software.
2. **Programs are mathematical objects**; – they are expressed in formal language that have formal semantics which makes it possible for programs to be treated as mathematical theories.

Parts to formal methods

The main parts to formal methods are:

1. Formal specification.

Using mathematics to specify the desired properties of a computer system.

2. Formal verification.
Using mathematics to prove that a computer system satisfies its specification. To which many would add:
3. Automated programming.
Automating the process of program generation.

Advantage of formal method

- Formal Method forces the System Analyst and Designer to think carefully about the specification as it enforce proper engineering approach using discrete mathematics.
- Formal Method forces the System Analyst and Designer to see all the different possible states for any given variables and functions thus will avoid many faults and therefore reduces the bugs and errors from the design stage onward.

Disadvantage of formal method

- Formal Method requires the person to know how to apply discrete mathematics. It will obviously slow down the analysis and design stage resources and time therefore also the cost of the project.
- There are too many different formal methods and most of them are not compatible with each other.
- Formal methods do not guarantee that a specification is complete. For each variable and function, it just forces the System Analyst and Designer to view the specification from a different perspectives but it does not guarantee that variable and functions will not be left out.

Stages in Formal Method

The main stages in Formal Methods can be divided into the following:
1. **Formal Specification**
This is where normal system specification is use and translated using a formal language into a formal specification. There are basically two type of formal language; Model Oriented (VDM, Z, etc) and Properties Oriented (Algebraic Logic, Temporal Logic, etc). This is the cheapest way to handle formal method.
The formal specification generally does the following process.
1. Get user requirement usually from the specification written in the natural language.
2. Clarify the requirement using mathematical approach. This is to remove all ambiguous, incomplete and inconsistent statement.

3. After statements are clearly identified. Then find all assumptions (Things that must be in place before something can happen) that is state or not stated within the clarified requirement.
4. Then expose every possible logic defect (fault) or omission in the clarified requirement.
5. Identify what are the exceptions (bad things) that will arise if the defects are not corrected.
6. Find a way to test for all the possible each exception. Only when you can test for an exception can you be able to stop that exception from happening.

2. **Formal Proof**
This level studies the formal specification and retrieves the goals of the formal specific. Then fixed rules are created and with these rules step by step instructions are listed to achieve the specified goals. This is relatively cheaper but there are more task steps.

3. **Model Checking**
This level studies the formal specification and formal proof deliverables to make sure that the system or software contains ALL possible properties to be able to handle all possible scenarios that could happen for a given specification. This stage is beginning to be more expensive.

4. **Abstraction**
This level uses mathematical and physical models to create a prototype of the entire system for simulation. This prototype is use to focus on the properties and characteristic of the system. This is the most expensive formal method.

Formal Methods Model
The **formal methods model** is concerned with the application of a mathematical technique to design and implement the software. This model lays the foundation for developing a complex system and supporting the program development. The formal methods used during the development process provide a mechanism for eliminating problems, which are difficult to overcome using other software process models. The software engineer creates formal specifications for this model. These methods minimize specification errors and this result in fewer errors when the user begins using the system.

Formal methods comprise *formal specification* using mathematics to specify the desired properties of the system. Formal specification is expressed in a language whose syntax and semantics are formally defined. This language comprises a

syntax that defines specific notation used for specification representation; semantic, which uses objects to describe the system; and a set of relations, which uses rules to indicate the objects for satisfying the specification.

Generally, the formal method comprises two approaches, namely model-based and property based.

The **property-based or axiomatic specification:** This approach focuses on the properties that the proposed system is to satisfy, and there is no intention to produce an abstract model of the system. The required properties and behaviour of the system are stated in mathematical notation. It describes the operations performed on the system. In addition, it describes the relationship that exists among these operations. A property-based specification consists of two parts: signatures, which determine the syntax of operations and an equation, which defines the semantics of the operations through a set of equations known as **axioms.** The property-oriented approach has the advantage that the implementer is not constrained to a particular choice of implementation, and the only constraint is that the implementation must satisfy the stipulated properties. The emphasis is on specifying the required properties of the system, and implementation issues are avoided. The properties are typically stated using mathematical logic (or higher-order logics). Mechanized theorem-proving techniques may be employed to prove results. One potential problem with the axiomatic approach is that the properties specified may not be realized in any implementation.

The **model-based specification:** The model-oriented approach to specification is based on mathematical models, where a model is a simplification or abstraction of the real world that contains only the essential details. For example, the model of an aircraft will not include the colour of the aircraft, and the objective would be to model the aerodynamics of the aircraft. There are many models employed in the physical world, such as meteorological models that allow weather forecasts to be given. The importance of models is that they serve to explain the behaviour of a particular entity and may also be used to predict future behaviour. Models may vary in their ability to explain aspects of the entity under study. One model may be good at explaining some aspects of the behaviour, whereas another model might be good at explaining other aspects. A good model is chosen as a representation of the real world and is referred to whenever there are questions in relation to the aspect of the real world. A model-based specification comprises a definition of the set of states of the system and definitions of the legal operations performed on the system to indicate how these legal operations change the current state.

Advantages and Disadvantages of Formal Methods Model

Various advantages and disadvantages associated with a formal method model are listed in Table.

Advantages	Disadvantages
Discovers ambiguity, incompleteness, and inconsistency in the software. Offers defect-free software. Incrementally grows in effective solution after each iteration. This model does not involve high complexity rate. Formal specification language semantics verify self-consistency.	Time consuming and expensive. Difficult to use this model as a communication mechanism for non technical personnel. Extensive training is required since only few developers have the essential knowledge to implement this model.

Systematic method for designing program

Design methods are procedures, techniques, aids, or tools for designing. They offer a number of different kinds of activities that a designer might use within an overall design process. The development of design methods has been closely associated with prescriptions for a systematic process of designing. These process models usually comprise a number of phases or stages, beginning with a statement or recognition of a problem or a need for a new design and culminating in a finalized solution proposal.

Models in SDLC

The models include:

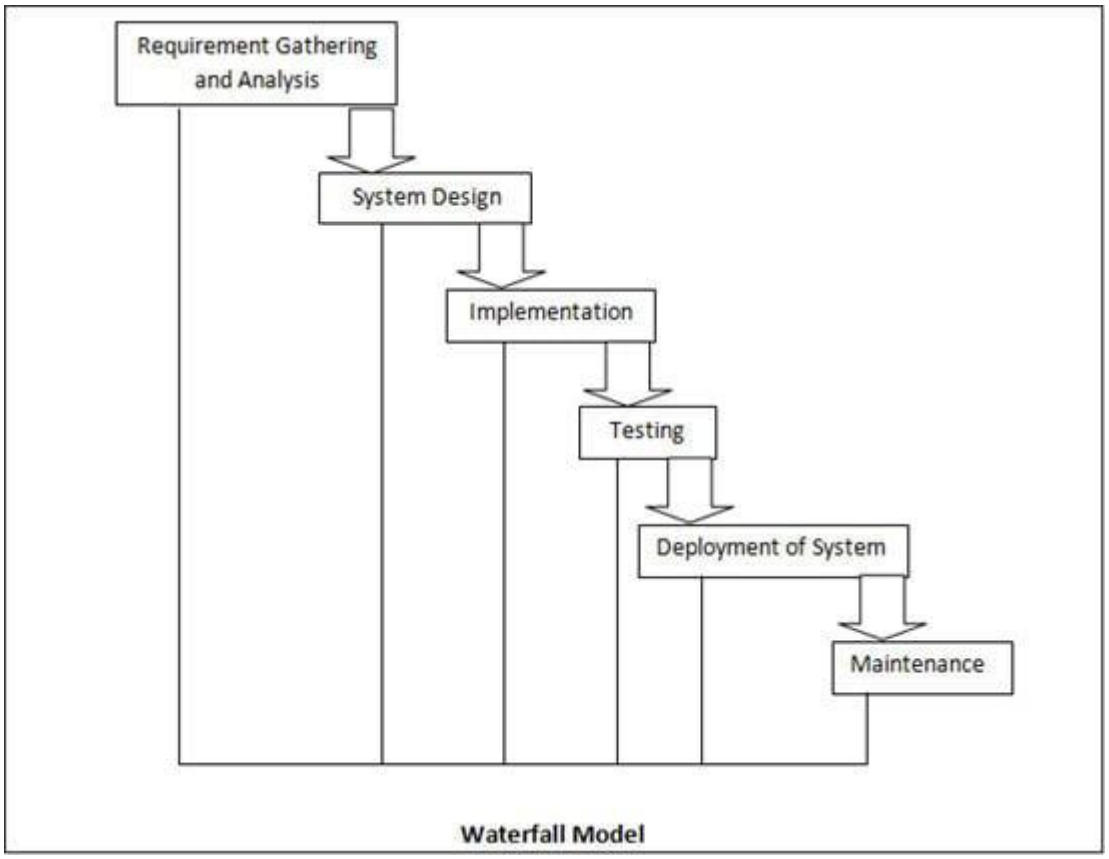
- Waterfall model
- V model
- Agile model
- Spiral model
- RAD

1) Waterfall Model

This model represents multiple stages or processes in a sequential manner that flows progressively downward.

This approach is useful when requirements are well known, technology is understood and the resources with required expertise are available.

Waterfall model is defined by the following stages:



- **Requirement Gathering and Analysis:** Capture and analyze all the requirements and make sure whether they are testable or not.
- **System Design:** Create and document design based on requirement analysis. Define the hardware and software requirements.
- **Implementation:** Create robust code for components as per the design and integrate them.
- **System Testing:** Integrated components form a whole system, this phase is performed to ensure whether the system is working as per the requirements, tracking and reporting the testing progress.
- **System Deployment:** Make sure if system is stable with zero bugs, all test criteria had been met, ensure Environment Setup etc.
- **System Maintenance:** Makes sure if the application is working efficiently as per the requirement with the suitable environment. In case a defect is found then that should be fixed and deployed (updated) in the environment.

Advantages of Waterfall Model:

- Simple and easy to understand.
- Easy to manage as each phase has its own specific deliverables.
- Overlapping of stages is avoided.
- Good for small projects.

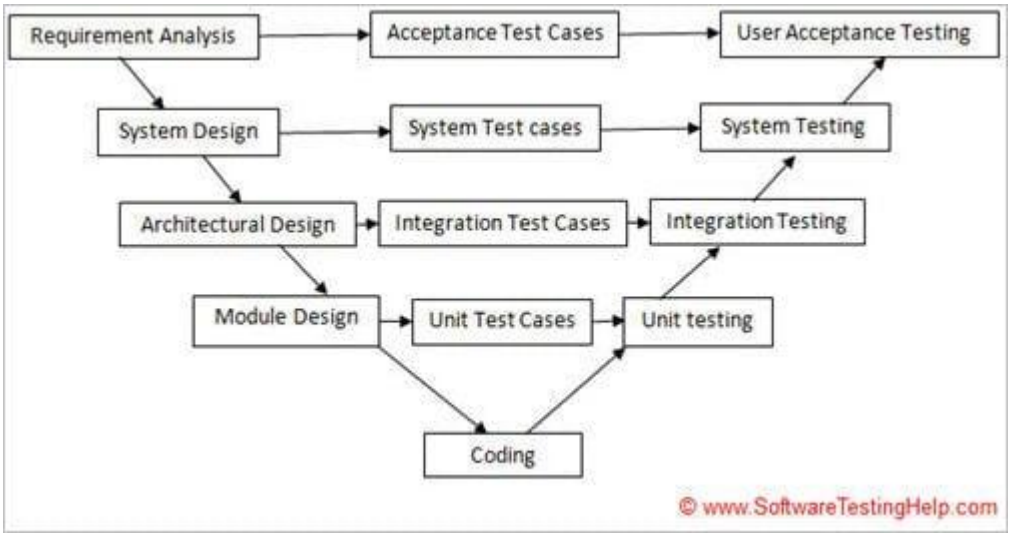
Disadvantages of Waterfall Model:

- Increase in the amount of risk and uncertainty.
- Once entered into Testing phase, cannot change anything in the previous stages **e.g** Design and Coding etc.
- Not good for complex and large projects.
- Not suitable where the requirements keep changing.

2) V Model

V Model is an *extension of Waterfall Model* where the process execution takes place in a sequential style in V-Shape and is also known as Verification and Validation Model. In this approach, there exists a directly associated testing phase in every single phase of the development cycle.

It has been proven beneficial and cost-efficient than the waterfall model as the testing is performed at each development phase rather than at the end of the development cycle.



V Model is classified into 3 Phases.

- Verification Phase
- Coding Phase
- Validation Phase

a) Verification Phase:

- **Business Requirement Analysis:** Communicate with the customer to understand their expectations and requirements.
- **System Design:** Design complete system and its components along with the hardware and software requirements.
- **Architectural Design:** In this phase architectural specifications are captured. This also known as high-level Design.
- **Module Design:** This is also known as Low-Level Design, Detailed internal design for all the specified system modules.

b) Coding Phase:

This phase contains actual coding phase in the development lifecycle. Programming languages should be chosen based on the system and architectural design specified in the previous phase technology platform. Coding is performed according to the standards and guidelines that are pre-defined.

c) Validation Phase:

- **Unit Testing:** Performed on an individual module to eliminate the bugs at the early stage.
- **Integration Testing:** Performed to test the communication between different modules in the system.
- **System Testing:** System Testing is performed on a system as a whole.
- **Acceptance Testing:** This is associated with the business requirements. It is performed in a user environment from the user's point of view.

Advantages of V model

- Simple, easy to use and understand.
- Overlapping is avoided as phases are executed one at a time.
- Easy to manage and suitable for small projects.

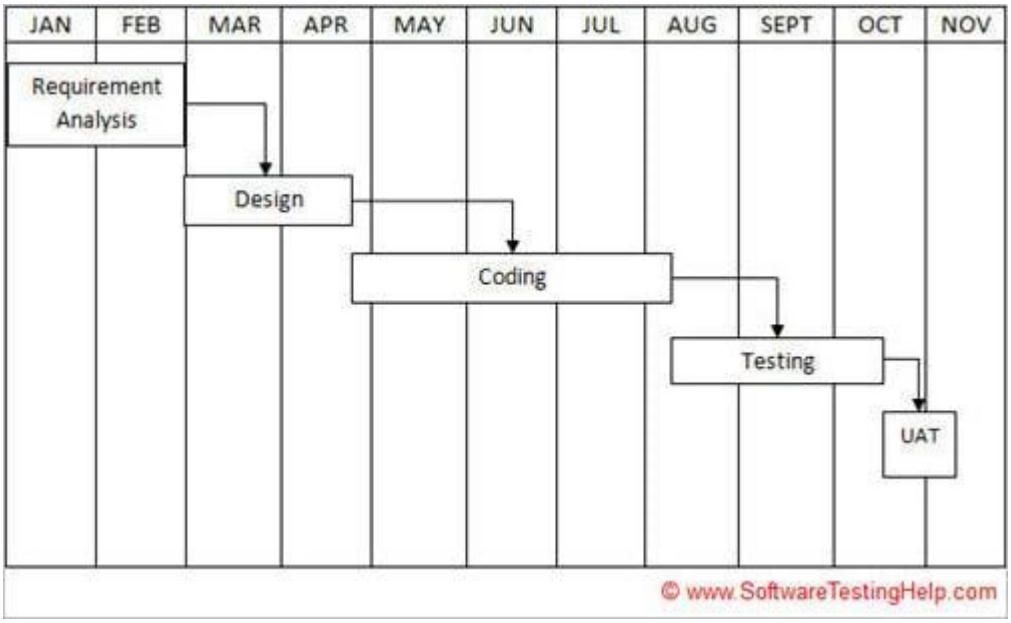
Disadvantages of V Model are more or less similar to the disadvantages of Waterfall model.

3) Agile Model

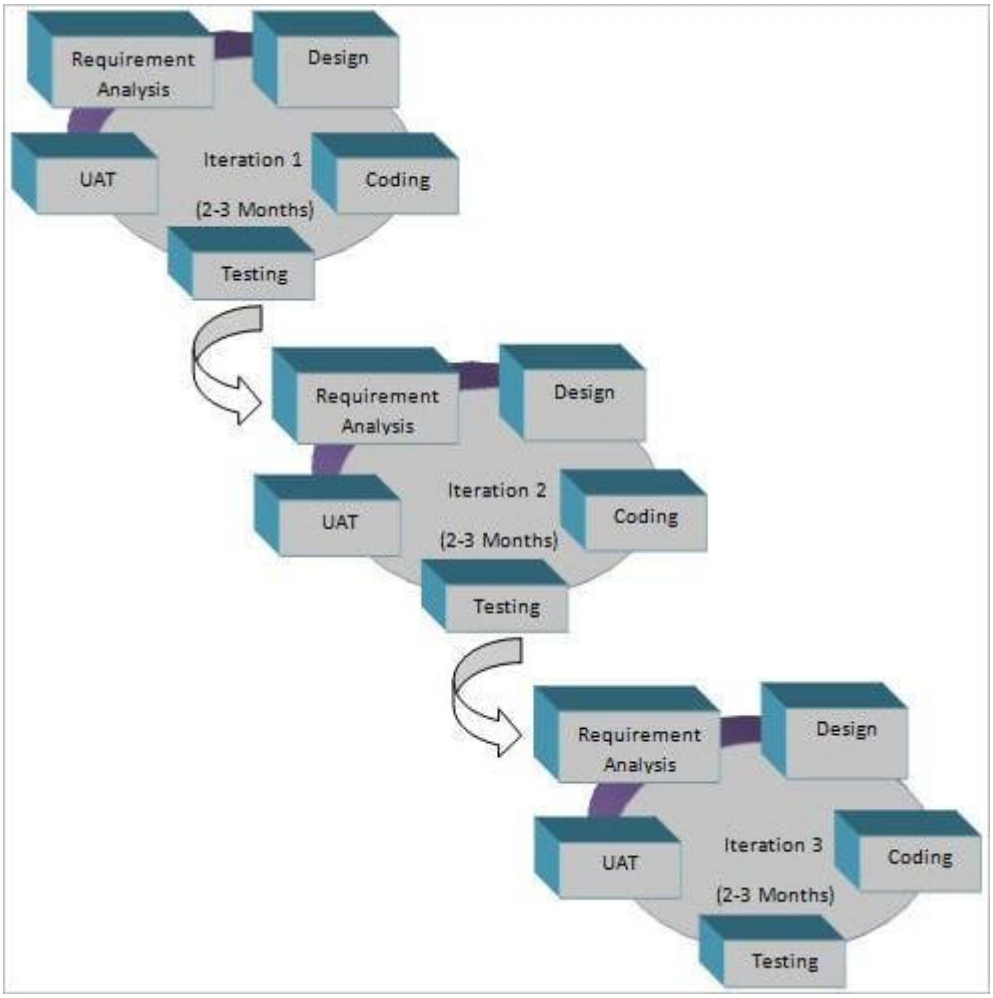
Agile Model shows an iterative and incremental approach. This approach breaks the product into small incremental units to provide iterations. Then each iteration involves steps like Planning, Requirement Analysis, Design, Coding, Unit Testing, Acceptance Testing etc.

This approach also allows continuous interaction with the customer for their feedback and corrections in the requirements at regular intervals.

The following diagram will help you to understand Agile Model approach more precisely:



Following image will show the iteration cycle in Agile Model:



Advantages of Agile model:

- A realistic approach to software development.
- Promotes teamwork.
- Eliminates mismatch between requirements and test cases.
- Rapid and requires minimum amount of resources.
- Suitable for large and long-term projects.
- Good for changing requirements.
- Easy to manage.

Disadvantages of Agile model:

- Not suitable for complex projects.
- Requires heavy amount of interaction with the customer which may cause delay.
- Misguidance of requirements may cause the incorrect development of the software product.
- Increased maintainability risk.
- Handover to another team may be quite challenging.

4) Spiral Model

The spiral model incorporates iterative development approach along with the systematic approach of the waterfall model. It is similar to the incremental model and emphasis on Risk Analysis.

Spiral Model has four stages:

- Planning Phase
- Risk Analysis
- Engineering Phase
- Evaluation Phase

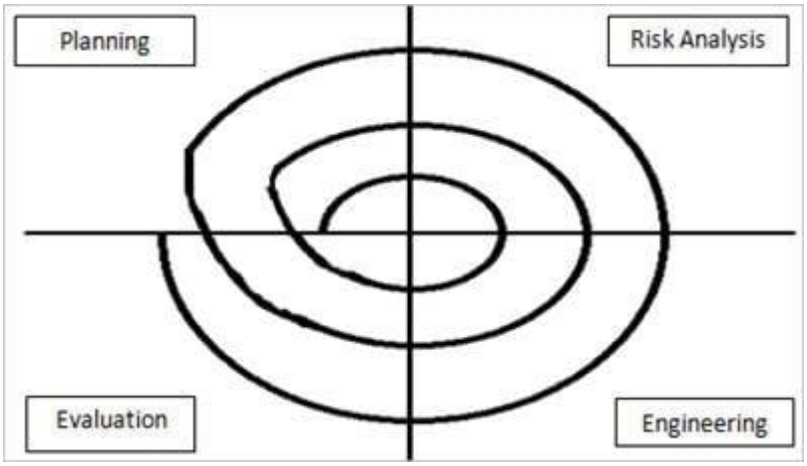
1) Planning Phase: In this phase, the requirements are gathered and reviewed to finalize the test case.

2) Risk Analysis: This stage includes identifying, monitoring and estimating management risks. Requirements are analyzed to identify the risks using techniques like brainstorming, walkthrough etc.

3) Engineering Phase: In this phase, the software is developed and tested at the end.

4) Evaluation Phase: This is the last stage where a customer evaluates the output of a project and gives their feedback for either next spiral or approval.

Pictorial representation of Spiral Model:



When to use Spiral model:

- For high-risk projects.
- When the requirements are complex.
- If a project is large.
- Have sufficient amount of time for getting user's feedback for the next spiral.
- Requires significant changes due to research and exploration.
- Users are not sure of their needs.

Advantages of Spiral Model:

- Avoidance of risk as it involves a high amount of risk analysis.
- Rapid Development.
- Changes in requirements are accommodated easily.
- Requirements can be acquired more accurately.

Disadvantages of Spiral model:

- Complex management.
- Not suitable for small projects.
- May involve no. of spirals(indefinite).
- Costly.
- Requires high amount of risk analysis and expertise for their project's success.

5) RAD Model

Rapid Application Development (RAD) is a type of incremental model. In this approach, components are developed in parallel.

This is a rapid approach and it can give a fast product to the customer to provide feedback.

Phases in RAD are as follows:

- **Business Modeling:** Identifies vital information and its flow between various business channels.

- **Data Modeling:** Information gathered in the previous stage is used to define data objects required for the business.
- **Process modeling:** Data objects are converted to get business objective and flow of information.
- **Application Generation:** In this phase, automation tools are used to convert the process model into actual code.
- **Testing and Turnover:** Tests all the components of a system, hence overall testing time is reduced.

Advantages of RAD Model:

- Progress can be measured.
- Reduces development time.
- Increased reusability.
- Quick initial reviews.
- Enhances customer feedback.

Disadvantages of RAD Model:

- Requires high skilled resources.
- High-cost estimation.
- Not applicable for cheaper projects.
- High dependency on modeling skills.
- Only a modularized system can be built using RAD.

Systematic method for coding program

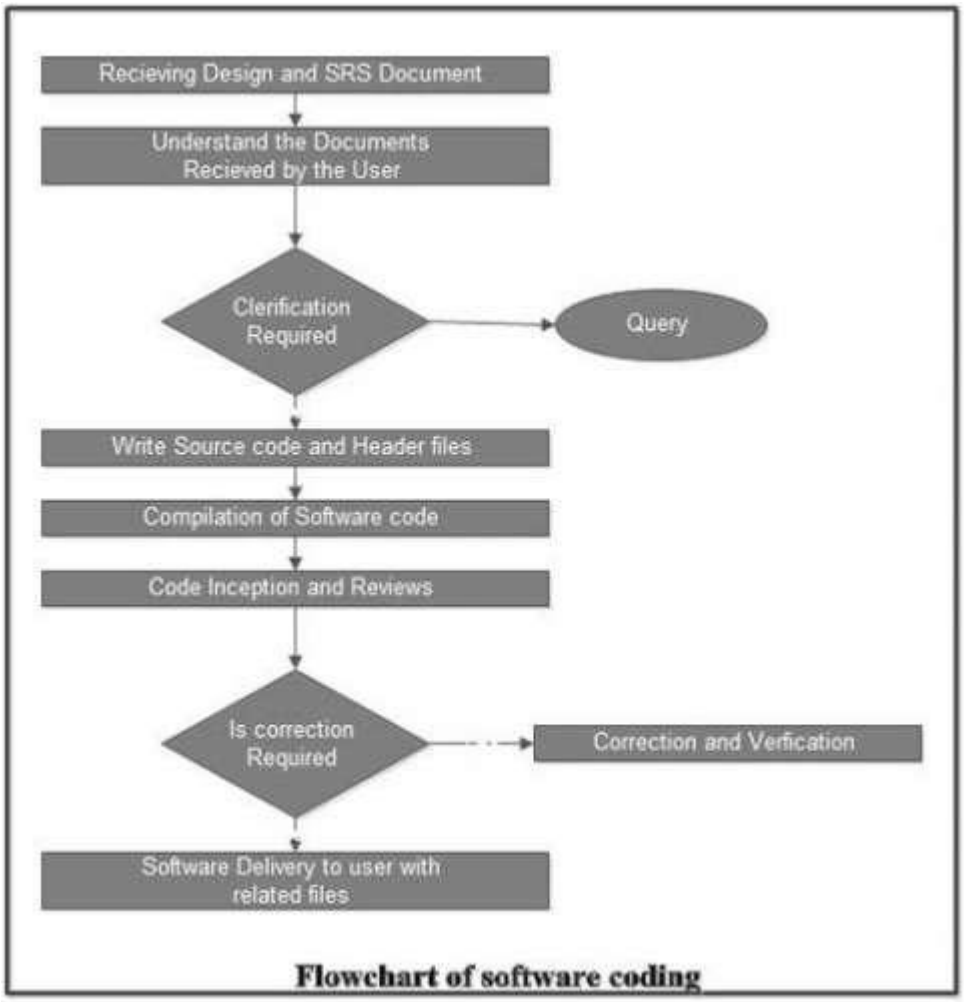
Coding methodology refers to a set of well-documented procedures and guidelines used in the analysis, design, and implementation of programs. Coding methodology includes a diagrammatic notation for documenting the results of the procedure. It also includes an objective set (ideally quantified) of criteria for determining whether the results of the procedure are of the desired quality. The steps to use coding' methodology are listed below.

1. The software development team begins its work by reviewing and understanding the design and requirements specification documents. These documents are essential for understanding user requirements and creating a framework for the software code.
2. In case the software development team is unable to understand user requirements correctly and further clarification is required, the queries are sent back to the user. In addition, the software development team also returns the requirements that are understood by them.
3. After the requirements are clearly understood by the software development team, the design and specifications are implemented in source code, supporting files, and the header files. Note that while writing the software code, the coding style

guidelines should be followed. In some cases, there may be a proposal of change in hardware or software specifications. However, the requests for change are implemented only after the approval of the user.

4. When the software code is completely written, it is compiled along with other required files.
5. Code inspection and reviews are conducted after the compilation. These methods are used to correct and verify errors in the software code.
6. Software testing is carried out to detect and correct errors in each module of the software code.
7. After the software code is tested, the software is delivered to the user along with the relevant code files, header files, and documentation files.

In Software Coding Process further change and clarifications are required in the design or SRS, the software development team raises a query, which is sent to the user with the document containing what the software development team understood from the documents sent by the user. Changes are made only when the user has a positive response to the queries raised by the software development team.



Flowchart of software coding

Systematic method for testing program

Testing is an essential part of the Software Development Process. A robust and stable software product can be delivered with the use of standard testing methodologies that will help to predict the timeline of the software system. A software application may turn even more complex with a large number of platforms and devices. More importantly, it is required to ensure whether they meet the specified requirements and can be efficiently installed and operated on the user's machine or not.

Testing Methodologies

Methodologies can be considered as the set of testing mechanisms used in software development lifecycle from Unit Testing to System Testing. Selecting an appropriate testing methodology is considered to be the core of the testing process. Basically, there are 3 testing methodologies which are used for testing. They are White Box Testing, Black Box Testing, and Grey Box Testing. These are also called as ***Testing Techniques***. Each of the testing technique is briefed below for your better understanding.

1) White Box Testing:

White box testing technique is used to examine the program structure and business logic, it validates the code or program of an application. It is also called as *Clear Box Testing, Glass Box Testing or Open Box Testing*.

White Box Testing Techniques include:

- **Statement Coverage:** Examines all the programming statements.
- **Branch Coverage:** Series of running tests to ensure if all the branches are tested.
- **Path Coverage:** Tests all the possible paths to cover each statement and branch.

2) Black Box Testing:

Black Box testing method is used to test the functionality of an application based on the requirement specification. Unlike White Box Testing it does not focus on internal structure/code of the application.

Black Box Techniques include:

- Boundary Value analysis
- Equivalence Partitioning(Equivalence Class Partitioning)
- Decision Tables
- Domain Tests
- State Models
- Exploratory Testing(Requires less preparation and also helps to find the defects quickly).

3) Grey Box Testing:

This method of testing is performed with less information about the internal structure of an application. Generally, this is performed like Black Box Testing only but for some critical areas of application, White Box Testing is used.

Difference between Testing Methodologies & Testing Strategies

The answer to this is not much complex as there is a simple difference between both.

Testing Methodologies are the methods or approaches to testing that includes from Unit testing through System Testing.

Testing Strategies is an overview of the key issues that occur in the testing process and is to be taken into consideration by the project manager, a team of developers and testers.

The above-discussed Software Testing Methods are used to implement *n* number of testing strategies.

Some of them are listed below:

1) Unit Testing:

- Focuses on very small functional units.
- The simplest way to check smallest units for isolation.
- Generally performed by developers.

2) Integration Testing:

- This is the next step to be performed on the developer's side.
- Provide mechanism to test interaction, inter-operation, and communication between the different modules of software

3) Functional Testing:

It is used to check the functionalities of a software system i.e. output to the given input.

4) Regression Testing:

Checks if the bug fixing has happened at one place so that the complex functionalities should not cause any change in another core area.

5) System Testing:

- Testing all the integrated modules as a collective system.
- Combines multiple features into end-to-end scenarios.

6) Performance Testing:

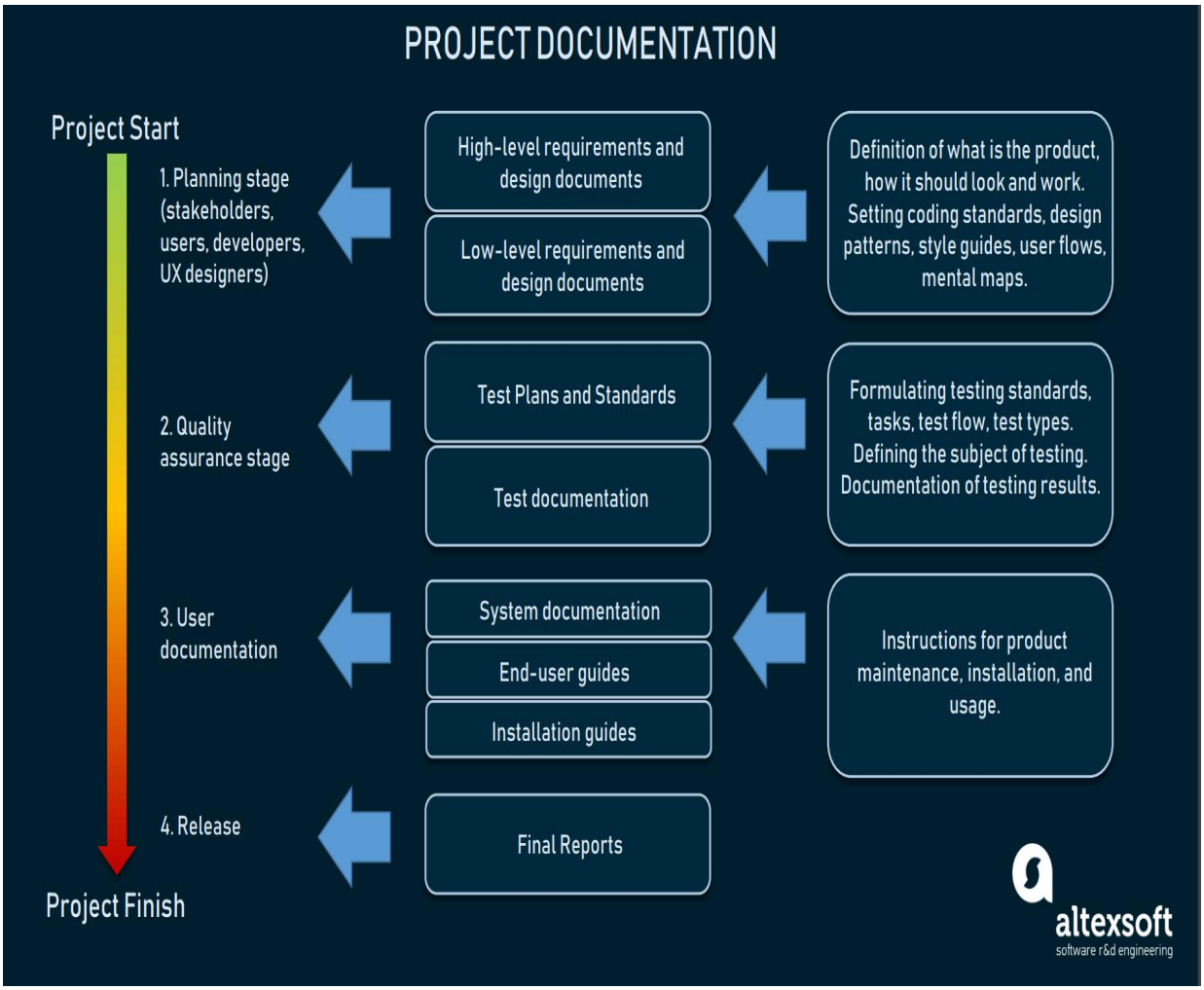
Tests the performance of the application in critical situations like transferring big sized file, concurrent users access to the system, configuration failure etc.

7) Acceptance Testing:

- Generally Final level of testing where software product is examined as users perspective by testers
- The result of this step is subjective and takes a little to find exact issue

Systematic method for documenting program

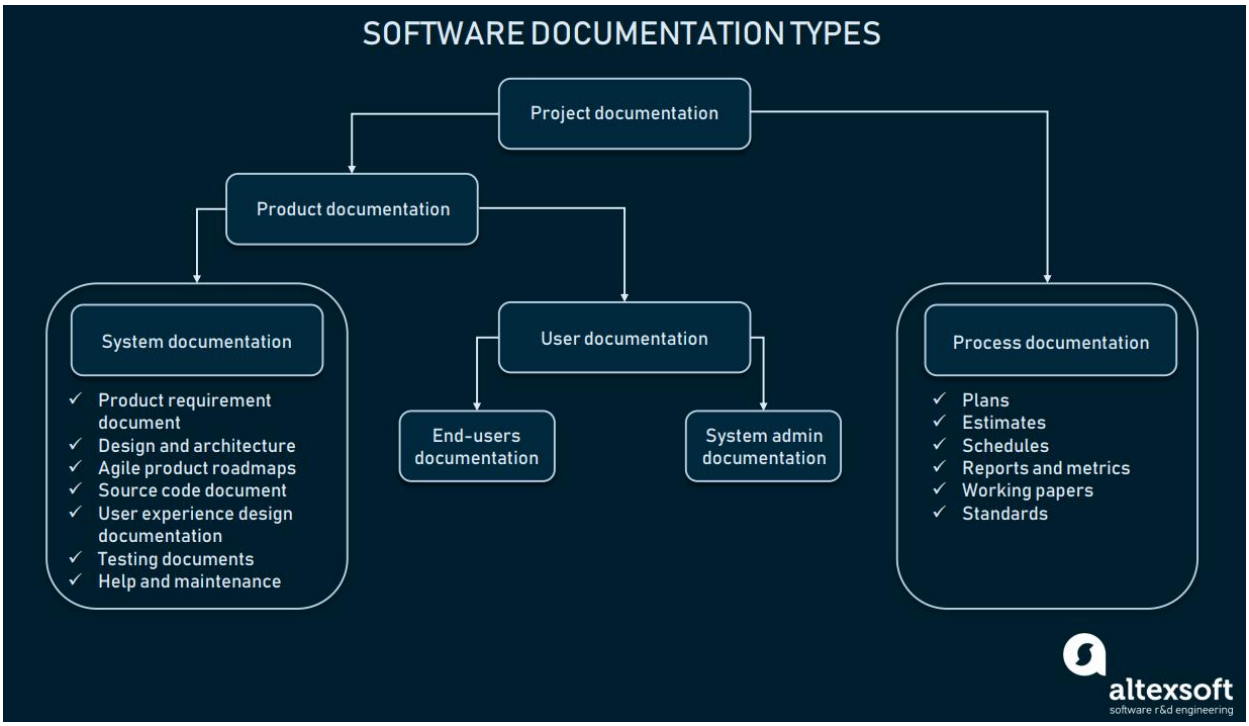
Technical documentation in software engineering is the umbrella term that encompasses all written documents and materials dealing with software product development. All software development products, whether created by a small team or a large corporation, require some related documentation. And different types of documents are created through the whole software development lifecycle (SDLC). Documentation exists to explain product functionality, unify project-related information, and allow for discussing all significant questions arising between stakeholders and developers.



Types of documentation

The main goal of effective documentation is to ensure that developers and stakeholders are headed in the same direction to accomplish the objectives of the project. To achieve them, plenty of documentation types exist.

Adhering to the following classifications.



All software documentation can be divided into two main categories:

- Product documentation
- Process documentation

Product documentation describes the product that is being developed and provides instructions on how to perform various tasks with it. In general, product documentation includes requirements, tech specifications, business logic, and manuals. There are two main types of product documentation:

- System documentation represents documents that describe the system itself and its parts. It includes requirements documents, design decisions, architecture descriptions, program source code, and FAQs.
- User documentation covers manuals that are mainly prepared for end-users of the product and system administrators. User documentation includes tutorials, user guides, troubleshooting manuals, installation, and reference manuals.

Process documentation represents all documents produced during development and maintenance that describe… well, the process. The common examples of process-related documents are standards, project documentation, such as project plans, test schedules, reports, meeting notes, or even business correspondence.

The main difference between process and product documentation is that the first one record the process of development and the second one describes the product that is being developed.

Formal specification, abstraction, modularity and reusability

Formal Specification

Formal Specification is the initial part of formal method that describes what the system must do without saying how it is to be done. It is totally language independent and focuses only on the abstract rather than detail logic.

A formal specification can serve as a single, reliable reference point for those who investigate the customer's needs, those who implement programs to satisfy and those needs, those who test the results, and those who write instruction manuals for the system.

Formal Specification in the SDLC

Two (2) things are very important during requirement gathering; the data and the process done on the data. Formal specification will be applied directly on these things.

If formal Specification is use during the Analysis and Design stage, there is no need to use them again. If formal specification is not used by the SA and SD then the only window of opportunity to apply it will be during the pre-development stage.

If no formal specification was ever applied, and it is applied in the testing stage, this will expose a lot of possible bugs not catered for and the problem will loop back to the design and possibly the analysis stage.

Analyst Stage

The main purpose of Analyze is to find the source of the problem. During this stage the System Analyst (SA) would collect all the data and processes (observation, document inspection, interview, etc…). The SA would then express the requirements into some form of diagrams such as DFD or Rich Picture, Use Cases and Case Diagram, etc. Then the SA will write a report (in English – common natural language) to indicate the source of the problem and some alternative solution. When studying the current system the SA could apply proposition and predicate to every client's statement thus translating them into mathematical equivalent. This will remove ambiguity and expose all possible hidden state of the process and data. Proof is then use to be sure if the statement given by the client is correct. The SA can also apply set theories and series to categories and view data from different perspective. This is very helpful when SA needs to understand how reports are generated.

By doing this the SA can be to a certain degree confident to cover all possible alternative to a given statement.

Design Stage

The main purpose of Design is to find create a specification for the selected solution. It should be stressed that, the specification will be use to built the

solution, thus a good specification will create a good software and a bad specification will create a bad software. During this stage the System Designer (SD) uses the analyze report to create the specification. The SD also expresses their specification into some form of diagrams sometimes similar to those used by the SA. Before creating the specification the SD could translate all the natural language found in the analyze report into mathematical equivalent (proposition and predicates) that will remove all ambiguity and uncertainty. Proof can be use here to ensure that every statement given is logically correct. Set theories and series are use to categories and view data from different perspective and to create relevant reports.

Then studying the mathematical form, the SD will be able to create the new system environment and also the solution that can cater for all possible scenario and state for each data and processes.

Development Stage

The main purpose of Development is to find built the solution base on the specification. During this stage the Senior Programmer (SP) will study the specification, create the relevant data structure, study the modules and delegate the programming team to develop the solution. To apply formal specification at this stage will be a bit late but a small window of opportunity still exists. During this stage the Senior Programmer (SP) will study the specification. The SP could translate all the natural language found in the specification into mathematical equivalent (proposition and predicates) that will remove all ambiguity and uncertainty. Proof can be use here to ensure that every statement given is logically correct. Set theories and series are use to categories and view data from different perspective and to create relevant reports. The SP can then verify that the specification is complete before starting out the development.

If there is any problem with the design, the SP will stop the development and return the specification to the SD for correction. Worst case scenario, the entire specification is drop and the system is reanalyzed.

Testing Stage

The main purpose of Testing is to make sure that the solution solves the problem found in the analysis and created base on the specification. Before this stage the (SA) will have already created the test plan and test case. In this stage the tester will then use the test plan and test case to execute the testing. To apply formal specification at this stage is really very late. During this stage the SA will revise the specification built during by the SD. The SA could then translate all the natural language found in the specification into mathematical equivalent (proposition and predicates) that will remove all ambiguity and uncertainty. Proof can be use here to ensure that every statement given is logically correct. Set theories and series are

use to categories and view data from different perspective and to create relevant reports. After studying the specification, then the SA can create a Test Plan that will cover all aspect of the system. Using what is learnt from the Formal specification, the SA will be able to create a test case for each test item to test out all the different exception. If there is any problem with the testing, the SA will stop the testing and return the specification to the SD for correction. Worst case scenario, the entire specification is drop and the system is reanalyzed.

Notice that this will not return to the development, because development only follows the specification created during the design stage. Formal Specification should not be use during the Implementation stage

Uses of Formal Specification

- **Formal specification involves investing more effort in the early phases** of software development. This reduces requirements errors as it forces a detailed analysis of the requirements
- **Incompleteness** and **inconsistencies** can be discovered and resolved. Hence, savings are made as the amount of rework due to requirements problems is reduced

Program Abstraction

Abstraction is the act of representing essential features without including the background details or explanations. In the computer science and software engineering domain, the abstraction principle is used to reduce complexity and allow efficient design and implementation of complex software systems. Some areas of software design and implementation where the abstraction principle is applied include programming languages (mainly in object-oriented programming languages), specification languages, control abstraction, data abstraction and the architecture of software systems.

Abstraction is one of the most important principles in object-oriented software engineering and is closely related to several other important concepts, including encapsulation, inheritance and polymorphism. Abstraction is applied in the process of identifying software artifacts (objects) to model the problem domain. It is the process of reducing these objects to their essence such that only the necessary elements are represented. Abstraction defines an object in terms of its properties, functionality, and interface (means of communicating with other objects).

These methods are used to reduce the complexity of the design and implementation process of software. In that process, the designers define abstract object actors that are able to perform work, change their state and communicate with other actors. The state of the object is encapsulated while the detailed data structures associated with the object are kept behind the scenes.

Types of Abstraction

Abstraction can be of two types, namely, *data abstraction* and *control abstraction.* **Data abstraction** means hiding the details about the data and **control abstraction** means hiding the implementation details. In object-oriented approach, one can abstract both data and functions. However, generally, the classes in OOP are defined in such a way that the data is hidden from the outside world and the functions form the public interface. That is, the functions of the class can be directly accessed by other functions outside the class and the hidden data can be accessed indirectly with the help of these functions.

Since, the internal details of the class are hidden from the outside world, thus, data abstraction ensures security of data by preventing it from accidental changes or manipulations by other parts of the program.

Abstraction layer

In computing, an abstraction layer or abstraction level is a way of hiding the working details of a subsystem, allowing the separation of concerns to facilitate interoperability and platform independence. Examples of software models that use layers of abstraction include the OSI model for network protocols, OpenGL and other graphics libraries.

In computer science, an abstraction layer is a generalization of a conceptual model or algorithm, away from any specific implementation. These generalizations arise from broad similarities that are best encapsulated by models that express similarities present in various specific implementations. The simplification provided by a good abstraction layer allows for easy reuse by distilling a useful concept or design pattern so that situations where it may be accurately applied can be quickly recognized.

A layer is considered to be on top of another if it depends on it. Every layer can exist without the layers above it, and requires the layers below it to function. Frequently abstraction layers can be composed into a hierarchy of abstraction levels. The OSI model comprises seven abstraction layers. Each layer of the model

encapsulates and addresses a different part of the needs of digital communications, thereby reducing the complexity of the associated engineering solutions.

Program Modularity

In software engineering, modularity refers to the extent to which a software/Web application may be divided into smaller modules. Software modularity indicates that the number of application modules are capable of serving a specified business domain. Software engineering modularity allows typical applications to be divided into modules, as well as integration with similar modules, which helps developers use prewritten code. Modules are divided based on functionality, and programmers are not involved with the functionalities of other modules. Thus, new functionalities may be easily programmed in separate modules. **Modularity** helps to have extensible, modifiable, portable, maintainable, reusable, understandable and flexible **software**. It allows new features to be seamlessly added when desired, and unwanted features to be removed, thus simplifying the user-facing view of the **software**. It allows several developers to work on different modules concurrently. It also allows to test modules independently. Furthermore, large projects become easier to monitor and to control. Modularity is successful because developers use prewritten code, which saves resources. Overall, modularity provides greater software development manageability.

Importance Of Modularity In Programming

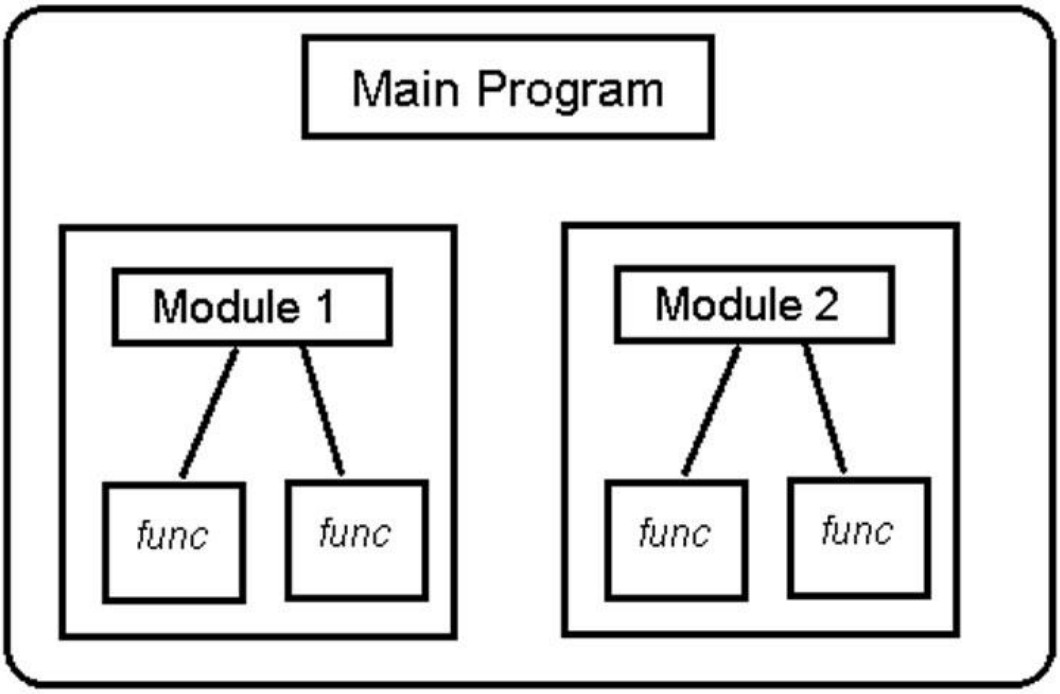
A major concern of software development using traditional procedural techniques is when it comes to big projects where complexities surround the large software project to be designed. In such projects that involves tens of thousands of lines of code, having a clear knowledge of what a segment of code does becomes more difficult.

To overcome the problem of a mix-up and the repeat of similar code in different stages of the program, modularity was designed to have codes designed in segments to do a specific function that can be referenced from any part of the program. Similar functions are put into a module. Since the modules are independent, the complete software is a collection of functional units or sub-programs to make the complete program.

Function of Modularity in Programming

Modularity in programming simply encourages the separation of the functionality in a program into distinct and independent units such that every unit has everything required for it to execute just that aspect of the program. The elements in the interface of a module can be detected in other modules of the program.

Modular programming today deals with high-level decomposition of an entire program's code into functional and independent units.



A simple diagram explaining modular programming structure.

The Importance Of Modular Programming

- **Ability To Reuse Code:** Modularity enables programmers to use the same code in the module over and over again by simply referencing the code to perform the specific action in the module at different locations of the program. Creating units or classes ensures this.
- **Easy Debugging:** A program with millions of lines of code will present a huge task if not in a module when debugging is required. You can imagine what it is like to search through such a huge environment to look for errors in a program. Having each task in its discrete module makes the process a lot easier to carry out. A faulty function can easily be checked for errors when in a module.
- **Simplicity And Ease Of Maintenance:** With modularity, the number of lines of codes should be reduced. Apart from handling the repetition of similar codes, modularity simplifies the whole process of developing a large project by having different programmers work on various aspects of the project.
- **Organization:** Modularization makes reading a program a lot better to do and understand. This makes the programmer better organized in his coding which helps when referencing. This also makes other programmers who

may want to go through the workings of the module find it much easier to read. Modularity thus improves readability through a well-organized design.

- **Modularity in Programming Encourages Teamwork and Collaboration:** Modular design in a program development improves collaborative efforts from different programmers. This builds the teamwork spirit that is required to complete a major project that requires thousands of lines of codes.
- **Modularity Is Reliable:** With all features of simplicity, ease of debugging, organization and other important factors, modularity can be described in one word, a 'reliable' technique to apply when dealing with software development

Software Reusability

Software reuse is the process of implementing or updating software systems using existing software components. A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses. In short, the development of a reuse process and repository produces a base of knowledge that improves in quality after every reuse, minimizing the amount of development work required for future projects, and ultimately reducing the risk of new projects that are based on repository knowledge.

Types of Reuse

Horizontal reuse

Horizontal reuse refers to software components used across a wide variety of applications. In terms of code assets, this includes the typically envisioned library of components, such as a linked list class, string manipulation routines, or graphical user interface (GUI) functions. Horizontal reuse can also refer to the use of a commercial off-the-shelf (COTS) or third-party application within a larger system, such as an e-mail package or a word processing program. A variety of software libraries and repositories containing this type of code and documentation exist today at various locations on the Internet.

Vertical Reuse

Vertical reuse, significantly untapped by the software community at large, but potentially very useful, has far reaching implications for current and future software development efforts. The basic idea is the reuse of system functional areas, or domains that can be used by a family of systems with similar functionality. The study and application of this idea has spawned another engineering discipline, called domain engineering. Domain engineering is a

comprehensive, iterative, life-cycle process that an organization uses to pursue strategic business objectives. It increases the productivity of application engineering projects through the standardization of a product family and an associated production process. Which brings us to application engineering, the domain engineering counterpart: Application engineering is the means by which a project creates a product to meet a customer's requirements. The form and structure of the application engineering activity are crafted by domain engineering so that each project working in a business area can leverage common knowledge and assets to deliver a high-quality product, tailored to the needs of its customer, with reduced cost and risk. Domain engineering focuses on the creation and maintenance of reuse repositories of functional areas, while application engineering makes use of those repositories to implement new products.