Requirement Engineering

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Let us see the process briefly -

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, and productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations

etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

There are several types of requirements that are commonly specified in this step, including:

- **Functional Requirements:** These describe what the software system should do. They specify the functionality that the system must provide, such as input validation, data storage, and user interface.
- **Non-Functional Requirements:** These describe how well the software system should do it. They specify the quality attributes of the system, such as performance, reliability, usability, and security.
- **Constraints:** These describe any limitations or restrictions that must be considered when developing the software system.
- **Acceptance Criteria:** These describe the conditions that must be met for the software system to be considered complete and ready for release.

In order to make the requirements specification clear, the requirements should be written in a natural language and use simple terms, avoiding technical jargon, and using a consistent format throughout the document. It is also important to use diagrams, models, and other visual aids to help communicate the requirements effectively.

Once the requirements are specified, they must be reviewed and validated by the stakeholders and development team to ensure that they are complete, consistent, and accurate.

Requirements verification and validation:

Verification: It refers to the set of tasks that ensures that the software correctly implements a specific function.

Validation: It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements. If requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework. The main steps for this process include:

- The requirements should be consistent with all the other requirements i.e no two requirements should conflict with each other.
- The requirements should be complete in every sense.
- The requirements should be practically achievable

Software Requirement Specification (SRS):

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Gathering requirements is an essential step in software development. Here are some steps you can follow to gather requirements for software development:

1. Identify stakeholders: Identify all stakeholders who will be affected by the software, such as customers, users, managers, and developers.
2. Define the purpose and scope of the software: Understand the objectives of the software, the business goals, and the problem you are trying to solve.
3. Collect user stories: Collect user stories, which are short descriptions of features that users want. These stories should capture what the user wants to do, why they want to do it, and what they expect to achieve.
4. Conduct interviews and surveys: Conduct interviews and surveys to gather information about user needs, preferences, and expectations. This information will help you understand the context in which the software will be used.
5. Create a prototype: Create a prototype of the software, which is a preliminary version of the software that can be tested and evaluated by stakeholders.
6. Prioritize requirements: Prioritize requirements based on their importance to the stakeholders and the business goals. This will help you determine which features to focus on first.
7. Document requirements: Document the requirements in a clear and concise manner, using tools such as use cases, user stories, and functional requirements.
8. Review and refine requirements: Review the requirements with stakeholders to ensure that they are complete, accurate, and meet their needs. Refine the requirements as necessary based on feedback.
9. Validate requirements: Validate the requirements by testing the software against them. This will ensure that the software meets the requirements and that the requirements are correct.

Functional Requirements: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Non-functional requirements: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

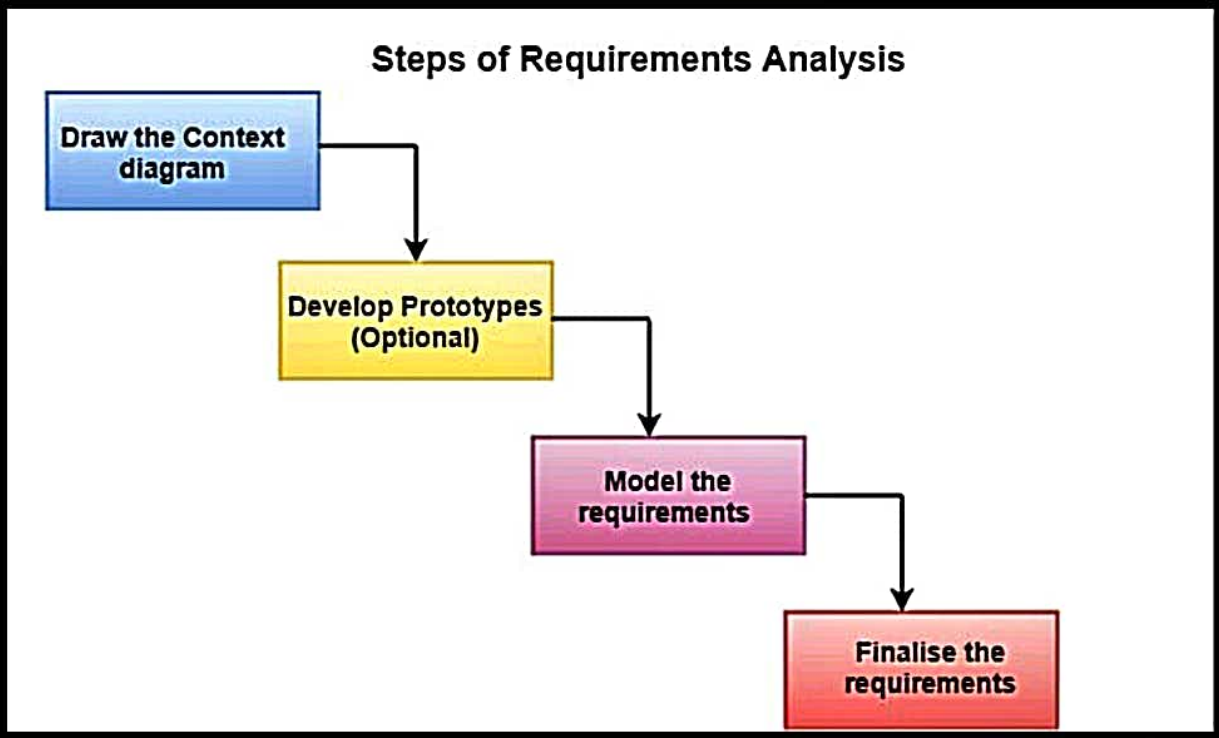
Following are the differences between Functional and Non-Functional Requirements:

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies "What should the software system do?"	It places constraints on "How should the software system fulfill the functional requirements?"
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.

Usually easy to define.	Usually more difficult to define.
Example **1)** Authentication of user whenever he/she logs into the system. **2)** System shutdown in case of a cyber attack. **3)** A Verification email is sent to user whenever he/she registers for the first time on some software system.	**Example** **1)** Emails should be sent with a latency of no greater than 12 hours from such an activity. **2)** The processing of each request should be done within 10 seconds **3)** The site should load in 3 seconds when the number of simultaneous users are > 10000

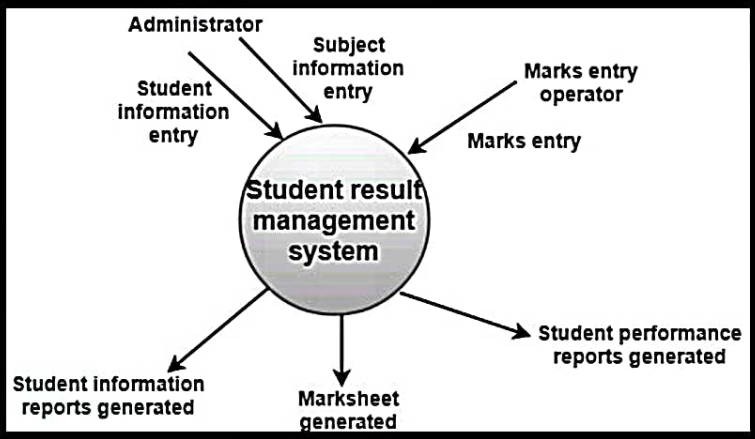
It is significant and essential activity after elicitation. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.

The various steps of requirement analysis are shown in fig:



(i) Draw the context diagram: The context diagram is a simple model that defines the

boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system. The context diagram of student result management system is given below:



(ii) Development of a Prototype (optional): One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.

We can use their feedback to modify the prototype until the customer is satisfied continuously. Hence, the prototype helps the client to visualize the proposed system and increase the understanding of the requirements. When developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision.

Some projects are developed for the general market. In such cases, the prototype should be shown to some representative sample of the population of potential purchasers. Even though a person who tries out a prototype may not buy the final system, but their feedback may allow us to make the product more attractive to others.

The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity.

(iii) Model the requirements: This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships between them. The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

(iv) Finalise the requirements: After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system. Now we finalize the analyzed requirements, and the next step is to document these requirements in a prescribed format.

Requirements verification and validation:

Verification: It refers to the set of tasks that ensures that the software correctly implements a specific function.

Validation: It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements. If requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework. The main steps for this process include:

- The requirements should be consistent with all the other requirements i.e no two requirements should conflict with each other.
- The requirements should be complete in every sense.
- The requirements should be practically achievable.

Reviews, buddy checks, making test cases, etc. are some of the methods used for this.

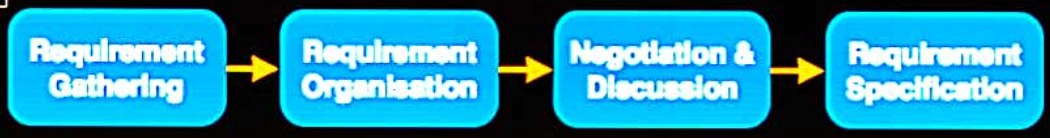
Requirements verification and validation (V&V) is the process of checking that the requirements for a software system are complete, consistent, and accurate, and that they meet the needs and expectations of the stakeholders. The goal of V&V is to ensure that the software system being developed meets the requirements and that it is developed on time, within budget, and to the required quality.

- Verification is the process of checking that the requirements are complete, consistent, and accurate. It involves reviewing the requirements to ensure that they are clear, testable, and free of errors and inconsistencies. This can include reviewing the requirements document, models, and diagrams, and holding meetings and walkthroughs with stakeholders.
- Validation is the process of checking that the requirements meet the needs and expectations of the stakeholders. It involves testing the requirements to ensure that they are valid and that the software system being developed will meet the needs of the stakeholders. This can include testing the software system through simulation, testing with prototypes, and testing with the final version of the software.
- V&V is an iterative process that occurs throughout the software development life cycle. It is important to involve stakeholders and the development team in the V&V process to ensure that the requirements are thoroughly reviewed and tested.

It's important to note that V&V is not a one-time process, but it should be integrated and continue throughout the software development process and even in the maintenance stage.

Requirement Elicitation Process

Requirement elicitation process can be depicted using the following diagram:



- **Requirements gathering** – The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements** – The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion** – If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation** – All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

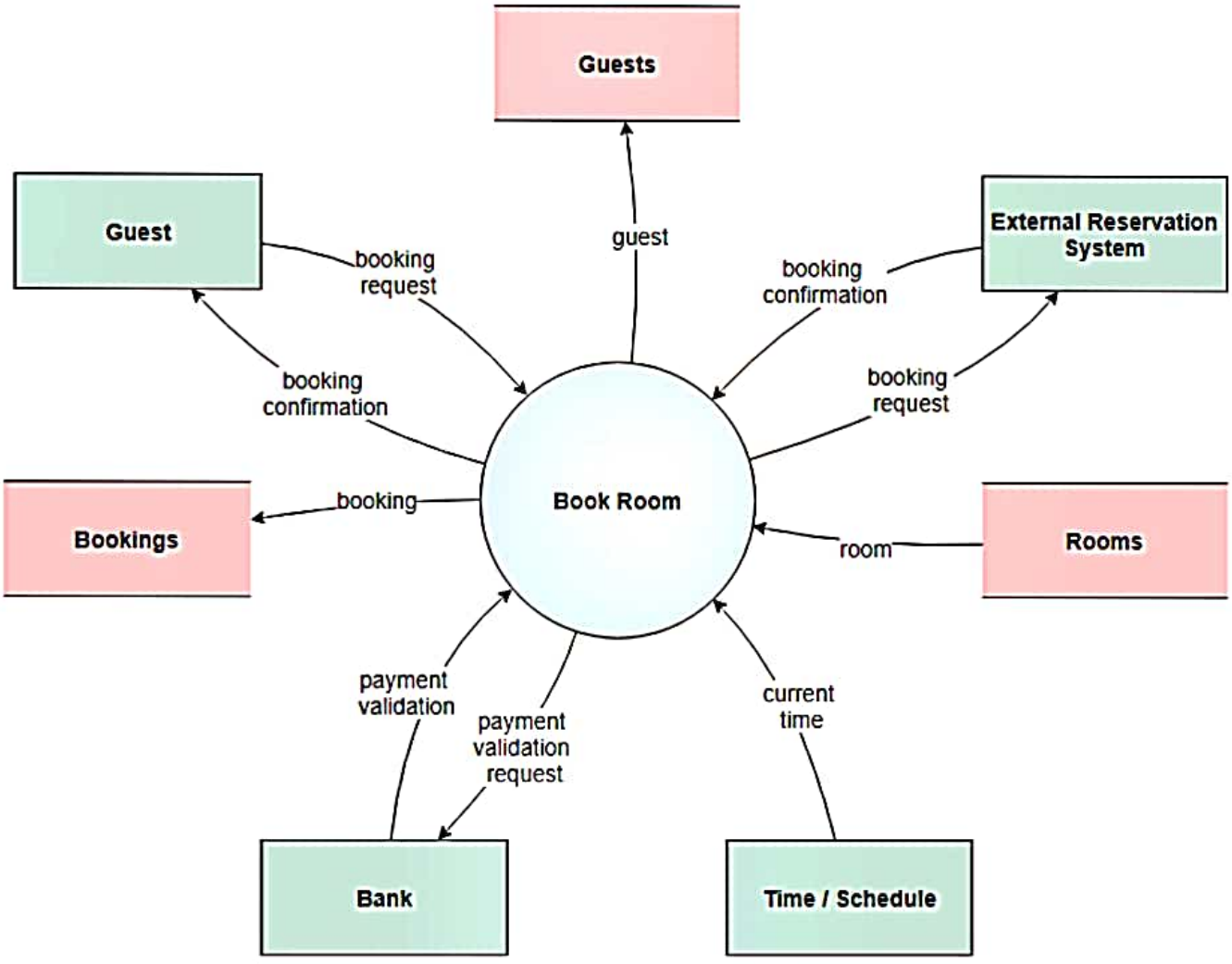
- **Requirements gathering** – The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements** – The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion** – If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation** – All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Example I: Context Diagram in a Hotel reservation system

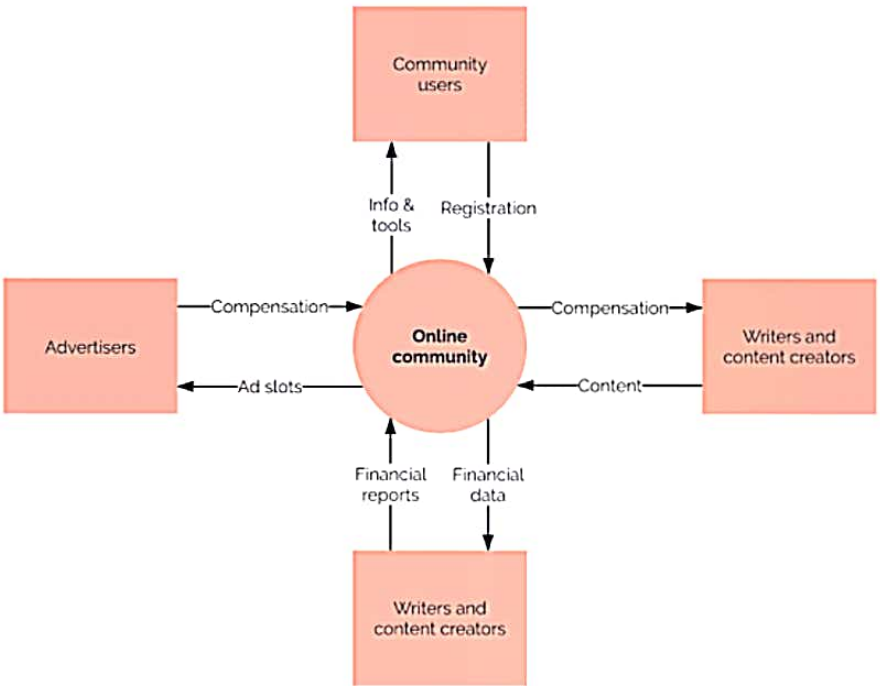
The graph below depicts the necessary components in a computerized system that distributes and stores the hotel information. It assists hotel managers in running their sales and online marketing activities, allowing them to update their room rates and vacant rooms and making them visible on their sales channels. These comprise both online and conventional travel agencies.



Also referred to as the Level O Data Flow Diagram, the **Context diagram** is the highest level in a Data Flow Diagram. It is a tool popular among Business Analysts who use it to understand the details and boundaries of the system to be designed in a project. It points out the flow of information between the system and external components.

It is made up of a **context bubble**, first drawn in the middle of the chart. It is usually a circle shape that represents a conceptual boundary that encloses a group of interconnected processes and activities of a project. The nitty-gritty details of the internal structure of a system are masked in a context diagram since it is strictly a high-level view of the system. This process is called information hiding.

A context diagram makes part of the requirements document in a project. Unlike other project diagrams, the **Context diagram** is not for use by the engineers/technicians but the project stakeholders. It, therefore, should be laid out in simple and understandable language for easy understanding of the items by the stakeholders when they analyze it.



Formal System Specification:

A formal software specification is **a statement expressed in a language whose vocabulary, syntax, and semantics are formally defined.** The need for a formal semantic definition means that the specification languages cannot be based on natural language; it must be based on mathematics.

It is **a mathematical method to specify a hardware or software system.** It verifies whether a specification is realisable, and verifies that an implementation satisfies its specification. It proves the properties of a system without necessarily running the system, etc.

The formal specification of a programming language is written in a form ready for machine execution or written using a formal mathematical notation. On the other hand, an informal specification can be expressed through a model such as UML or in natural language

These types of models can be categorized into the following specification paradigms:

- History-based specification. Behavior based on system histories. ...
- State-based specification. Behavior based on system states. ...
- Transition-based specification. ...
- Functional specification. ...
- Operational Specification.

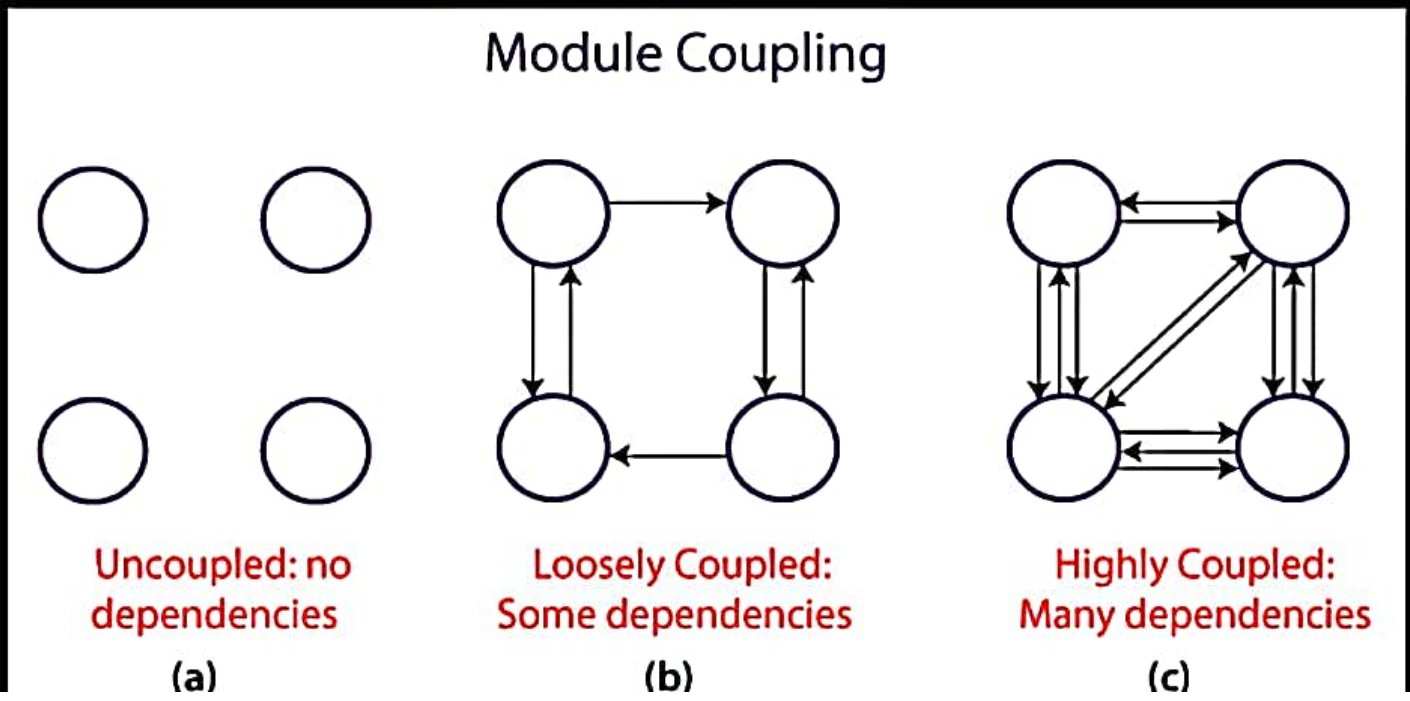
Benefits:

Formal specifications have several advantages over informal specifications. **They can be mathematically precise.** They tend to be more complete than informal specifications, because the formality tends to highlight any incompleteness, which might otherwise go unnoticed.

Cohesion and Coupling

In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. Uncoupled modules have no interdependence at all within them.

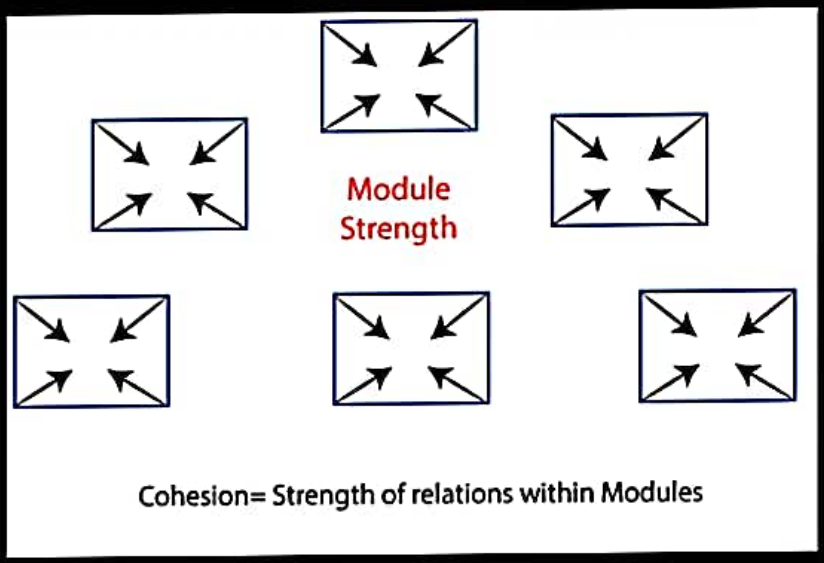
A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have more errors.



Cohesion

In computer programming, cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related.

Cohesion is an ordinal type of measurement and is generally described as "high cohesion" or "low cohesion."



Coupling	Cohesion
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.
Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.
Coupling shows the relative **independence** between the modules.	Cohesion shows the module's relative **functional** strength.
While creating, you should aim for low coupling, i.e., dependency among modules should be less.	While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.
In coupling, modules are linked to the other modules.	In cohesion, the module focuses on a single thing.