Functional and Non Functional Requirements

Requirements analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: *Functional* and *Non-functional requirements*.

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.

Functional Requirements	Non Functional Requirements
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

User Requirements

User requirements are statements, in a natural language plus diagrams, of what **services the system is expected to provide to system users and the constraints under which it must operate**. ... System requirements are more detailed descriptions of the software system's functions, services, and operational constraints.

The **user requirement(s) document** (URD) or **user requirement(s) specification** (URS) is a document usually used in software engineering that specifies what the user expects the software to be able to do.

Once the required information is completely gathered it is documented in a URD, which is meant to spell out exactly what the software must do and becomes part of the contractual agreement. A customer cannot demand features not in the URD, while the developer cannot claim the product is ready if it does not meet an item of the URD.

The URD can be used as a guide for planning cost, timetables, milestones, testing, etc. The explicit nature of the URD allows customers to show it to various stakeholders to make sure all necessary features are described.

Formulating a URD requires negotiation to determine what is technically and economically feasible. Preparing a URD is one of those skills that lies between a science and an art, requiring both software technical skills and interpersonal skills.

System Requirements

System requirements are the configuration that a system must have in order for a hardware or software application to run smoothly and efficiently**. Failure to meet these requirements can result in installation problems or performance problems. ... System requirements are also known as minimum system requirements**

System requirements is a statement that identifies the functionality that is needed by a system in order to satisfy the customer's requirements. [1] System requirements are a broad and also narrow subject that could be implemented to many items. Whether discussing the system requirements for certain computers, software, or the business processes from a broad view point. Also, taking it down to the exact hardware or coding that runs the software. System requirements are the most effective way of meeting the user needs and reducing the cost of implementation. [2] System requirements could cause a company to save a lot of money and time, and also can cause a company to waste money and time. They are the first and foremost important part of any project, because if the system requirements are not fulfilled, than the project is not complete.

Interface Specification

A user interface specification (UI specification) is a document that captures the details of the software user interface into a written document. The specification covers all possible actions that an end user may perform and all visual, auditory and other interaction elements.

Interface specifications provide the standardized mechanism in which subsystems can effectively communicate with each other and enable them to operate as independent modules that, when collectively implemented, support the entire CM technical reference model.

Software Requirements Specification (SRS) Document

A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfill all stakeholders (business, users) needs.

A typical SRS includes:

- A purpose
- An overall description
- Specific requirements

The best SRS documents define how the software will interact when embedded in hardware — or when connected to other software. Good SRS documents also account for real-life users.

Why Use an SRS Document?

A software requirements specification is the basis for your entire project. It lays the framework that every team involved in development will follow.

It's used to provide critical information to multiple teams — development, quality assurance, operations, and maintenance. This keeps everyone on the same page.

Using the SRS helps to ensure requirements are fulfilled. And it can also help you make decisions about your product's lifecycle — for instance, when to retire a feature.

Writing an SRS can also minimize overall development time and costs. Embedded development teams especially benefit from using an SRS.

***Software* Requirements Specification vs. *System* Requirements Specification**

A **software requirements specification (SRS)** includes in-depth descriptions of the software that will be developed.

A **system requirements specification (SyRS)** collects information on the requirements for a system.

"Software" and "system" are sometimes used interchangeably as SRS. But, a software requirement specification provides greater detail than a system requirements specification.

How to Write an SRS Document

Writing an SRS document is important. But it isn't always easy to do.

Here are five steps you can follow to write an effective SRS document.

1. Create an Outline (Or Use an SRS Template)

Your first step is to create an outline for your software requirements specification. This may be something you create yourself. Or you may use an existing SRS template.

If you're creating this yourself, here's what your outline might look like:

1. Introduction

1.1 Purpose

1.2 Intended Audience

1.3 Intended Use

1.4 Scope

1.5 Definitions and Acronyms

2. Overall Description

2.1 User Needs

2.2 Assumptions and Dependencies

3. System Features and Requirements

    3.1 Functional Requirements

    3.2 External Interface Requirements

    3.3 System Features

    3.4 Nonfunctional Requirements

Feasibility Study

Feasibility Study in Software Engineering is a study to evaluate feasibility of proposed project or system. Feasibility study is one of stage among important four stages of Software Project Management Process. As name suggests feasibility study is the feasibility analysis or it is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view. Feasibility study is carried out based on many purposes to analyze whether software product will be right in terms of development, implantation, contribution of project to the organization etc.

Types of Feasibility Study :
The feasibility study mainly concentrates on below five mentioned areas. Among these Economic Feasibility Study is most important part of the feasibility analysis and Legal Feasibility Study is less considered feasibility analysis.

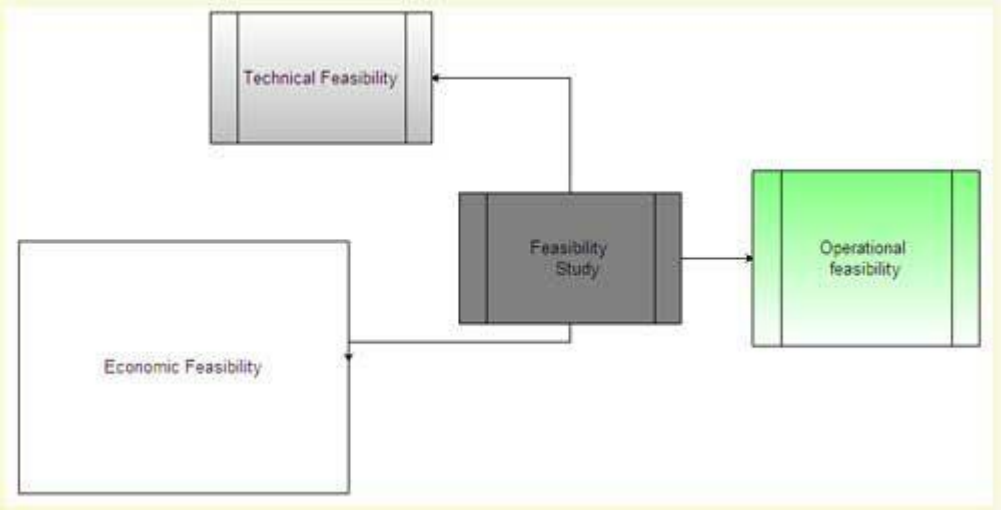
Feasibility is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software. Information such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance are considered during the feasibility study. The objective of the feasibility study is to establish the reasons for developing the software that is acceptable to users, adaptable to change and conformable to established standards. Various other objectives of feasibility study are listed below.

• To analyze whether the software will meet organizational requirements.

• To determine whether the software can be implemented using the current technology and within the specified budget and schedule.

Types of Feasibility

The feasibility study mainly concentrates on below five mentioned areas. Among these Economic Feasibility Study is most important part of the feasibility analysis and Legal Feasibility Study is less considered feasibility analysis.



1. **Technical Feasibility –**
   In Technical Feasibility current resources both hardware software along with required technology are analyzed/assessed to develop project. This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development. Along with this, feasibility study also analyzes technical skills and

capabilities of technical team, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology etc.

2. **Operational Feasibility –**
   In Operational Feasibility degree of providing service to requirements is analyzed along with how much easy product will be to operate and maintenance after deployment. Along with this other operational scopes are determining usability of product, Determining suggested solution by software development team is acceptable or not etc.

3. **Economic Feasibility –**
   In Economic Feasibility study cost and benefit of the project is analyzed. Means under this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on. After that it is analyzed whether project will be beneficial in terms of finance for organization or not.

4. **Legal Feasibility –**
   In Legal Feasibility study project is analyzed in legality point of view. This includes analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc. Overall it can be said that Legal Feasibility Study is study to know if proposed project conform legal and ethical requirements.

5. **Schedule Feasibility –**
   In Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can't be completed on time.

Feasibility Study Process :
The below steps are carried out during entire feasibility analysis.

1. Information assessment
2. Information collection
3. Report writing
4. General information

Need of Feasibility Study :
Feasibility study is so important stage of Software Project Management Process as after completion of feasibility study it gives a conclusion of whether to go ahead with proposed project as it is practically feasible or to stop proposed project here as it is not right/feasible to develop or to think/analyze about proposed project again.
Along with this Feasibility study helps in identifying risk factors involved in developing and deploying system and planning for risk analysis also narrows the business alternatives and

enhance success rate analyzing different parameters associated with proposed project development

Feasibility Study Process

Feasibility study comprises the following steps.

• **Information assessment:** Identifies information about whether the system helps in achieving the objectives of the organization. It also verifies that the system can be implemented using new technology and within the budget and whether the system can be integrated with the existing system.

• **Information collection:** Specifies the sources from where information about software can be obtained. Generally, these sources include users (who will operate the software), organization (where the software will be used), and the software development team (which understands user requirements and knows how to fulfill them in software).

• **Report writing:** Uses a feasibility report, which is the conclusion of the feasibility study by the software development team. It includes the recommendations whether the software development should continue. This report may also include information about changes in the software scope, budget, and schedule and suggestions of any requirements in the system.

• **General information:** Describes the purpose and scope of feasibility study. It also describes system overview, project references, acronyms and abbreviations, and points of contact to be used. **System overview** provides description about the name of the organization responsible for the software development, system name or title, system category, operational status, and so on. **Project references** provide a list of the references used to prepare this document such as documents relating to the project or previously developed documents that are related to the project. **Acronyms and abbreviations** provide a list of the terms that are used in this document along with their meanings. **Points of contact** provide a list of points of organizational contact with users for information and coordination. For example, users require assistance to solve problems (such as troubleshooting) and collect information such as contact number, e-mail address, and so on.

Management summary: Provides the following information.

• **Environment:** Identifies the individuals responsible for software development. It provides information about input and output requirements, processing requirements of the software and the interaction of the software with other software. It also identifies system security requirements and the system's processing requirements

• **Current functional procedures:** Describes the current functional procedures of the existing system, whether automated or manual. It also includes the data-flow of the current system and the number of team members required to operate and maintain the software.

• **Functional objective:** Provides information about functions of the system such as new services, increased capacity, and so on.

• **Performance objective:** Provides information about performance objectives such as reduced staff and equipment costs, increased processing speeds of software, and improved controls.

• **Assumptions and constraints:** Provides information about assumptions and constraints such as operational life of the proposed software, financial constraints, changing hardware, software and operating environment, and availability of information and sources.

• **Methodology:** Describes the methods that are applied to evaluate the proposed software in order to reach a feasible alternative. These methods include survey, modeling, benchmarking, etc.

• **Evaluation criteria:** Identifies criteria such as cost, priority, development time, and ease of system use, which are applicable for the development process to determine the most suitable system option.

• **Recommendation:** Describes a recommendation for the proposed system. This includes the delays and acceptable risks.

• **Proposed software:** Describes the overall concept of the system as well as the procedure to be used to meet user requirements. **In** addition, it provides information about improvements, time and resource costs, and impacts. Improvements are performed to enhance the functionality and performance of the existing software. Time and resource costs include the costs associated with software development from its requirements to its maintenance and staff training. Impacts describe the possibility of future happenings and include various types of impacts as listed below.

• **Equipment impacts:** Determine new equipment requirements and changes to be made in the currently available equipment requirements.

• **Software impacts:** Specify any additions or modifications required in the existing software and supporting software to adapt to the proposed software.

• **Organizational impacts:** Describe any changes in organization, staff and skills requirement.

• **Operational impacts:** Describe effects on operations such as user-operating procedures, data processing, data entry procedures, and so on.

• **Developmental impacts:** Specify developmental impacts such as resources required to develop databases, resources required to develop and test the software, and specific activities to be performed by users during software development.

• **Security impacts:** Describe security factors that may influence the development, design, and continued operation of the proposed software.

• **Alternative systems:** Provide description of alternative systems, which are considered in a feasibility study. This also describes the reasons for choosing a particular alternative system to develop the proposed software and the reason for rejecting alternative systems.

It’s a process of interacting with customers and end-users to find out about the domain requirements, what services the system should provide, and the other constrains.

Domain requirements reflect the environment in which the system operates so, when we talk about an application domain we mean environments such as train operation, medical records, e-commerce etc.

It may also involve a different kinds of stockholders; end-users, managers, system engineers, test engineers, maintenance engineers, etc.

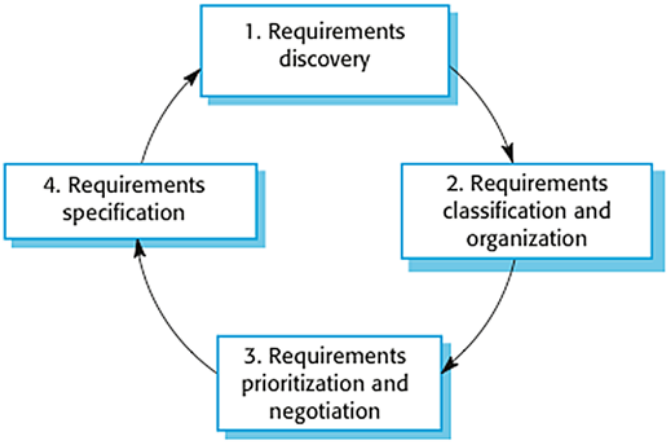
A stakeholder is anyone who has direct or indirect influence on the requirements.

The requirements elicitation and analysis has 4 main process

We typically start by gathering the requirements, this could be done through a general discussion or interviews with your stakeholders, also it may involve some graphical notation.

Then you organize the related requirements into sub components and prioritize them, and finally, you refine them by removing any ambiguous requirements that may raise from some conflicts.

Here are the 4 main process of requirements elicitation and analysis.



It shows that it's an iterative process with a feedback from each activity to another. The process cycle starts with requirements discovery and ends with the requirements document. The cycle ends when the requirements document is complete.

Requirements elicitation Methods:
There are a number of requirements elicitation methods. Few of them are listed below –

1. Interviews
2. Brainstorming Sessions
3. Facilitated Application Specification Technique (FAST)
4. Quality Function Deployment (QFD)
5. Use Case Approach

The success of an elicitation technique used depends on the maturity of the analyst, developers, users, and the customer involved.

1. Interviews:
Objective of conducting an interview is to understand the customer's expectations from the software.
It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.
Interviews maybe be open-ended or structured.

1. In open-ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.
2. In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

2. Brainstorming Sessions:

- It is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views
- A highly trained facilitator is required to handle group bias and group conflicts.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

3. Facilitated Application Specification Technique:
It's objective is to bridge the expectation gap – difference between what the

developers think they are supposed to build and what customers think they are going to get.
A team oriented approach is developed for requirements gathering.
Each attendee is asked to make a list of objects that are-

1. Part of the environment that surrounds the system
2. Produced by the system
3. Used by the system

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

4. Quality Function Deployment:
In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.
3 types of requirements are identified –

- **Normal requirements –**
  In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results, etc
- **Expected requirements –**
  These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorized access.
- **Exciting requirements –**
  It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when unauthorized access is detected, it should backup and shutdown all processes.

The major steps involved in this procedure are –

1. Identify all the stakeholders, eg. Users, developers, customers etc
2. List out all requirements from customer.
3. A value indicating degree of importance is assigned to each requirement.
4. In the end the final list of requirements is categorized as –
   - It is possible to achieve
   - It should be deferred and the reason for it
   - It is impossible to achieve and should be dropped off

5. Use Case Approach:
This technique combines text and pictures to provide a better understanding of the requirements.
The use cases describe the 'what', of a system and not 'how'. Hence, they only give a functional view of the system.
The components of the use case design includes three major things – Actor, Use cases, use case diagram.

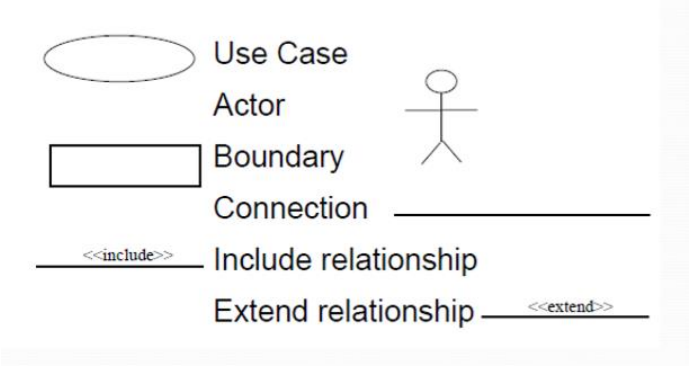
1. **Actor –**
   It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.
   - Primary actors – It requires assistance from the system to achieve a goal.
   - Secondary actor – It is an actor from which the system needs assistance.
2. **Use cases –**
   They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system.
3. **Use case diagram –**
   A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.
   - A stick figure is used to represent an actor.
   - An oval is used to represent a use case.
   - A line is used to represent a relationship between an actor and a use case.

Use Cases & Scenarios

The use cases and scenarios are two different techniques, but, usually they are used together.

Use cases identify interactions between the system and it's users or even other external systems (using graphical notations), while a scenario is a textual description of one or more of these interactions.

Use case involves some symbols to describe the system:



1. **Actors**: Are those who interact with the system; human or other systems
2. **Interaction (Use Case)**: It denotes the name of the interaction (verb). It's represented as a named ellipse.
3. **Connection**: Lines that links between the actors and the interactions.
4. **Include Relationship**: It denotes a connection between two interactions when an interaction is invoked by another. As an example, splitting a large interaction into several interactions.
5. **Exclude Relationship**: It denotes a connection between two interactions when you want to extend an interaction by adding an optional behavior, but you can use the main interaction on it's own without the extending interaction.

Now, we are going to use scenarios to describe the interactions in each use case textually. They should have a format and include the following:

1. A description of the initial situation.
2. A description of the flow of the events or interactions with the system
3. A description of the exceptions, or in other words, what can go wrong, and how it can be handled.
4. Any concurrent activities should be mentioned
5. A description of the final state.

Here is the example for a scenario for the use case example above.

Name	Course Registration
Actors	Student and University System
Description	It shows how a student can register for a course and view personal info
Pre-condition	The student is logged in
Post-condition	The student registered his/her course list for the semester.
Actions(Main Scenario)	1. Student will press on "Course Registration" from Home Page. 2. Select desired courses for the next semester. 3. Enter personal info. 4. Press on "Register". 5. Confirmation message upon success. 6. Student can view his/her personal info.
Exceptions	#3 User entered invalid input, thus, an error message will be displayed

Use case and scenarios are effective techniques for eliciting the requirements. But, because they focus on the interactions with the system, they aren't effective for eliciting high-level business, non-functional, or domain requirements.

Requirements validation is the process of checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation. We usually use requirements validation to check error at the initial phase of development as the error may increase excessive rework when detected later in the development process.
In the requirements validation process, we perform a different type of test to check the requirements mentioned in the Software Requirements Specification (SRS), these checks include:

- Completeness checks
- Consistency checks
- Validity checks
- Realism checks
- Ambiguity checks
- Verifiability

The output of requirements validation is the list of problems and agreed on actions of detected problems. The lists of problems indicate the problem detected during the process of requirement validation. The list of agreed action states the corrective action that should be taken to fix the detected problem.

There are several techniques which are used either individually or in conjunction with other techniques to check to check entire or part of the system:

1. **Test case generation:**
   Requirement mentioned in SRS document should be testable, the conducted tests reveal the error present in the requirement. It is generally believed that if the test is difficult or impossible to design than, this usually means that requirement will be difficult to implement and it should be reconsidered.
2. **Prototyping:**
   In this validation techniques the prototype of the system is presented before the end-user or customer, they experiment with the presented model and check if it meets their need. This type of model is generally used to collect feedback about the requirement of the user.
3. **Requirements Reviews:**
   In this approach, the SRS is carefully reviewed by a group of people including people from both the contractor organisations and the client side, the reviewer systematically analyses the document to check error and ambiguity.
4. **Automated Consistency Analysis:**
   This approach is used for automatic detection of an error, such as nondeterminism, missing cases, a type error, and circular definitions, in requirements specifications.

First, the requirement is structured in formal notation then CASE tool is used to check in-consistency of the system, The report of all inconsistencies is identified and corrective actions are taken.

5. **Walk-through:**
   A walkthrough does not have a formally defined procedure and does not require a differentiated role assignment.
   - Checking early whether the idea is feasible or not.
   - Obtaining the opinions and suggestion of other people.
   - Checking the approval of others and reaching an agreement.

Requirement Management

The Internet of Things (IoT) is changing not only the way products work, but their design and development. Products are continuously becoming more complex with more lines of code and additional software — some of which allow for even greater connectivity. With requirements management, you can overcome the complexity and interdependencies that exist in today's engineering lifecycles to streamline product development and accelerate deployment.

Issues in requirements management are often cited as major causes of project failures. Having inadequately defined requirements can result in scope creep, project delays, cost overruns, and poor product quality that does not meet customer needs and safety requirements.

Having a requirements management plan is critical to the success of a project because it enables engineering teams to control the scope and direct the product development lifecycle. Requirements management software provides the tools for you to execute that plan, helping to reduce costs, accelerate time to market and improve quality control.

Requirements management process
When looking for requirements management tools, there are a few key features to look for.

A typical requirements management process complements the systems engineering V model through these steps:

- Collect initial requirements from stakeholders
- Analyze requirements
- Define and record requirements
- Prioritize requirements
- Agree on and approve requirements
- Trace requirements to work items
- Query stakeholders after implementation on needed changes to requirements
- Utilize test management to verify and validate system requirements
- Assess impact of changes

- Revise requirements
- Document changes

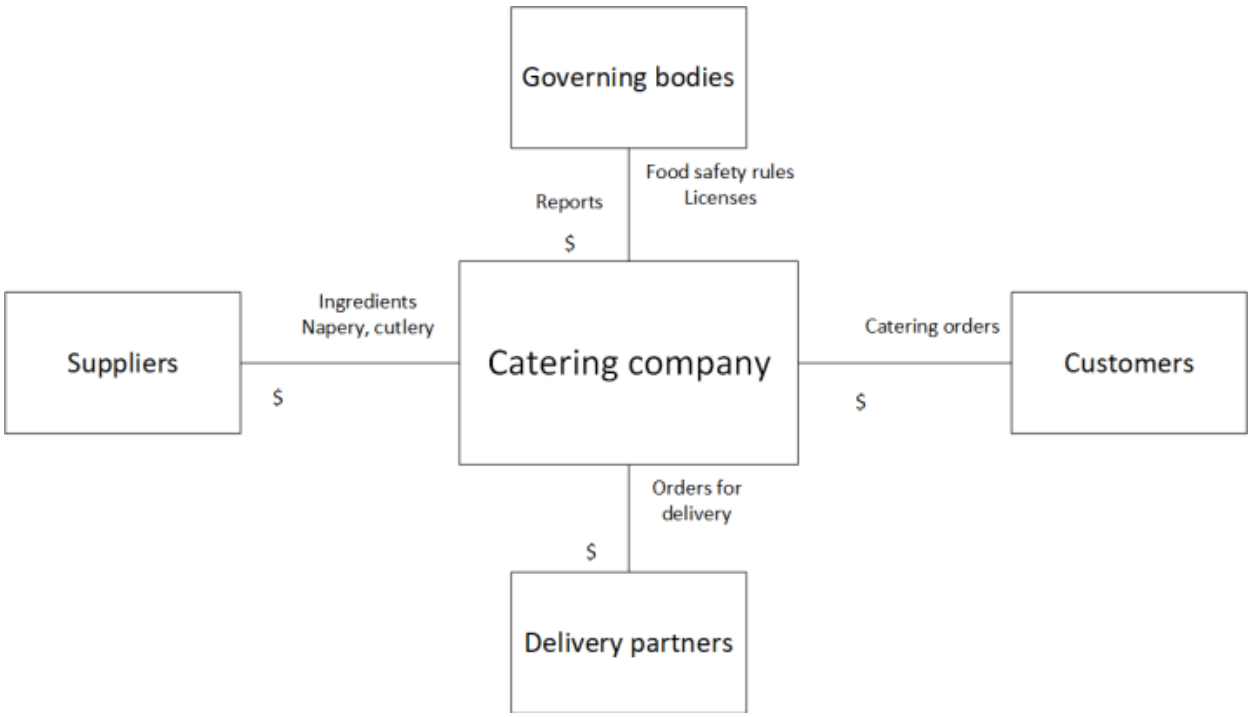
Requirements attributes
In order to be considered a “good” requirement, a requirement should have certain characteristics, which include being:

- Specific
- Testable
- Clear and concise
- Accurate
- Understandable
- Feasible and realistic
- Necessary

Context Model:

Context models are simple communication tools used to depict the context of a business, a system, or a process. The context is the environment in which the object of our interest exists. Context models capture how the central object interacts with its environment, be it exchanging data, physical objects, or funds.

Here is a simple context model of a catering company:



These models can be used to confirm project scope, identify potential impacts of changes, and start requirements discovery.

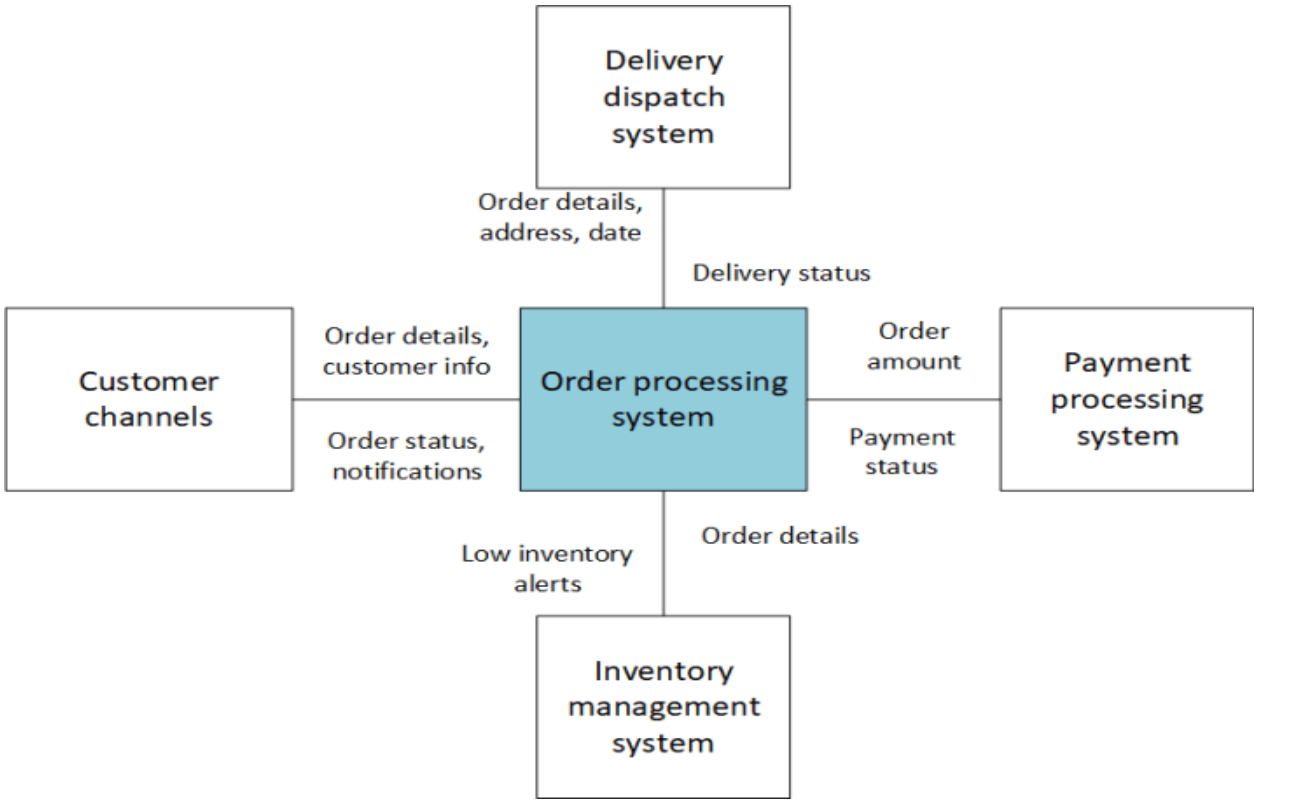
The key elements are:

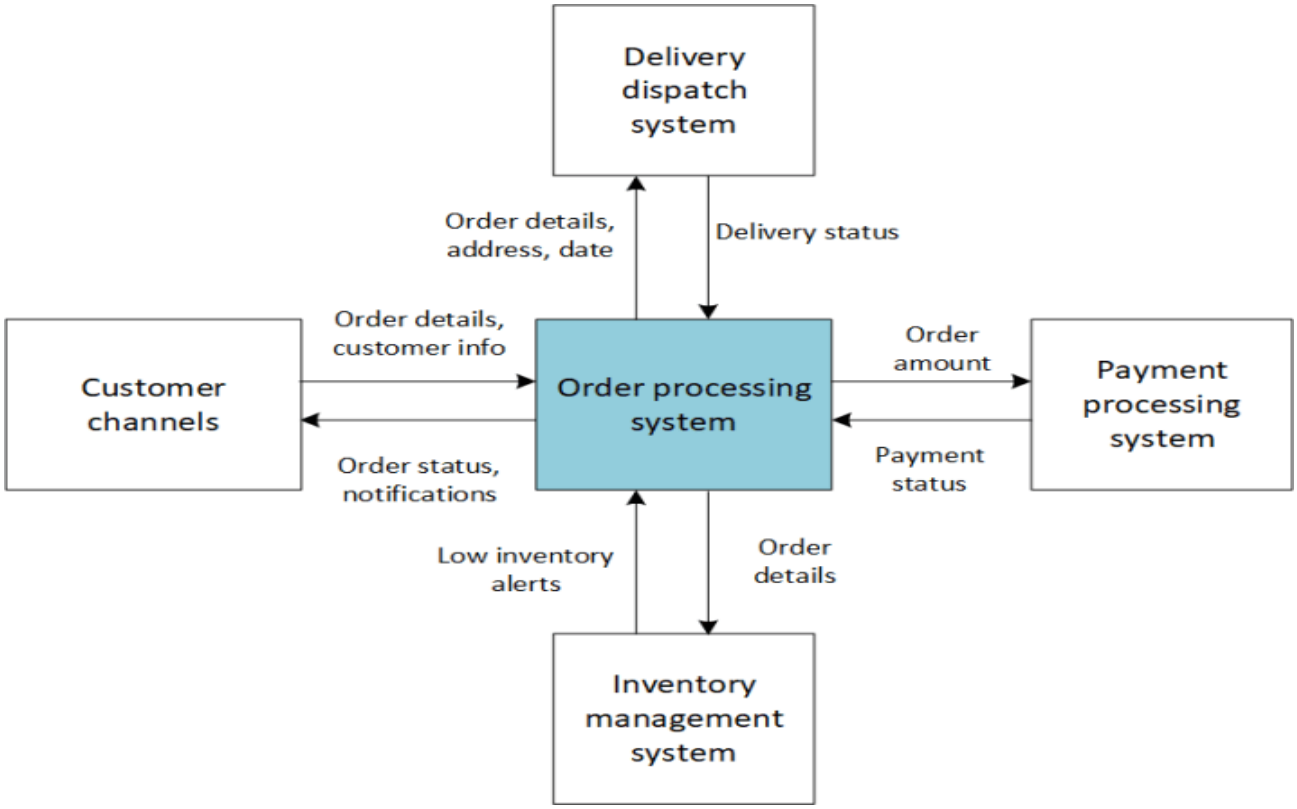
- The boxes: external entities that the main entity interacts with (organizations, departments, systems, or processes)
- The connectors: interactions between entities
- The flows: interactions are labelled to show the flow of data, physical objects, or funds between the entities.

That's it. Simple and powerful. This basic diagram can help discover gaps, missed impacts, interactions and requirements early in the project. And early is the best time, so that the scope of the project can be defined more accurately and realistically, and so that the analysis does not miss any important interactions.

Context diagrams are an excellent starting point for requirements planning and analysis. While customer journeys are a good approach when we need to take a customer perspective, it will not give us the full picture. As each organization interacts with its environment, industry, and value chain elements, we need to see it in the context to have a full picture.

The same logic can be applied to a context of a business process or a system. For example, imagine we want to build a solution for processing online orders. Without specifying what this solution would be, we can start with the context in which this solution will exist. This will be based on other solutions and applications that are already in place and will need to interact with the new solution:


Delivery dispatch system
Order details, address, date
Delivery status
Customer channels
Order details, customer info
Order status, notifications
Order processing system
Order amount
Payment status
Payment processing system
Low inventory alerts
Order details
Inventory management system


Delivery dispatch system
Order details, address, date
Delivery status
Customer channels
Order details, customer info
Order status, notifications
Order processing system
Order amount
Payment status
Payment processing system
Low inventory alerts
Order details
Inventory management system

Once you have creates a context model, you will find it remains useful through the lifecycle of an initiative:

Behavioral Model:

Overall behavior of a system can be fully understood by Behavioral model.

Behavioral Model is specially designed to make us understand behavior and factors that influence behavior of a System. Behavior of a system is explained and represented with the help of a diagram. This diagram is known as State Transition Diagram. It is a collection of states and events. It usually describes overall states that a system can have and events which are responsible for a change in state of a system.

Data Modelling

Data modelling is the process used to structure how data is stored, as well as modelling relationships within the data.

The goal is to **create a visual data map** that accurately describes the data structure, how **data will flow through the system** whilst **highlighting important data relationships**. This can involve the data input itself, the data infrastructure and output, whether that's predictive models, ML algorithms, AI or other products/services.

Database Data Modelling & Design Options

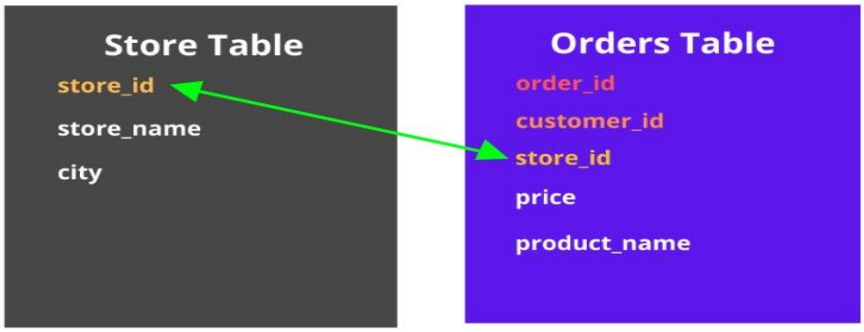
In modern software development there are three main ways of storing data:

Relational Databases (Entity models): These are made from tables (tabular data) with solutions including PostgreSQL and MySQL.

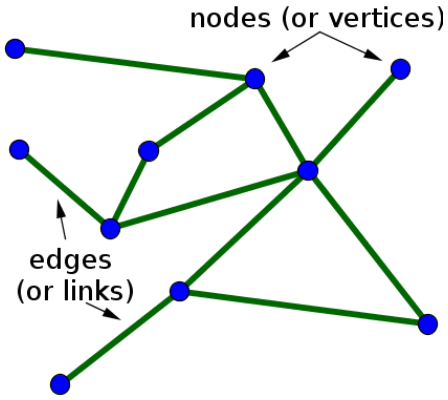
Let's imagine that we need to build an ordering system consisting of:

- Customers
- Orders
- Stores

The Store and Orders table could look like this:



- **Object Databases (NoSQL):** These are made up of **key, value** pairs and don't have a strict schema, several solutions include Firebase, MongoDB and Amazon DynamoDB.
- **Graph Databases (Tree):** Graph databases are often seen within social networks such as Facebook, a graph database is composed of nodes and edges, several solutions include Neo4j, Dgraph and Grakn.ai.



Object Model:

An object model is a logical interface, software or system that is modeled through the use of object-oriented techniques. It enables the creation of an architectural software or system model prior to development or programming.

An object model is part of the object-oriented programming (OOP) lifecycle.

An object model helps describe or define a software/system in terms of objects and classes. It defines the interfaces or interactions between different models, inheritance, encapsulation and other object-oriented interfaces and features.

Object model examples include:

- **Document Object Model (DOM)**: A set of objects that provides a modeled representation of dynamic HTML and XHTML-based Web pages
- **Component Object Model (COM)**: A proprietary Microsoft software architecture used to create software components

Structured Design:

Several representation techniques are used for structured analysis and structured style. the subsequent supports could be offered from CASE tools. A CASE tool ought to support one or a lot of structured analysis and style techniques. It ought to support effortlessly drawing analysis and style diagrams. The CASE tool ought to give simple navigation through the various levels and thru the look and analysis.
The tool should support completeness and consistency checking across the look and analysis and thru all levels of research hierarchy. Whenever it's potential, the system ought to forbid any inconsistent operation, however, it should be terribly tough to implement such a feature. Whenever there arises serious procedure load whereas consistency checking, it ought to be the potential to quickly disable consistency checking.

Code generation and CASE tools:
As way as code generation is anxious, the overall expectation of a CASE tool is kind of low. an affordable demand is a traceability from supply file to style knowledge. A lot of pragmatic supports expected from a CASE tool throughout the code generation section are the following:

- The CASE tool ought to support the generation of module skeletons or templates in one or a lot of fashionable languages. It ought to be the potential to incorporate copyright message, temporary description of the module, author name and therefore the date of creation in some selectable format.

- The tool ought to generate records, structures, category definition mechanically from the contents of the info wordbook in one or a lot of fashionable languages
- It ought to generate database tables for on-line database management systems.
- The tool ought to generate code for the programme from epitome definition for X window and MS window based mostly applications.

Test case generation CASE tool:

The CASE tool for action at code generation ought to have the subsequent features:

- It ought to support each style and demand testing.
- It ought to generally take a look at set reports in ASCII format which might be directly foreign into the test set up document.