geeksforgeeks_articles_on_srs

https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/?ref=lbp

https://www.geeksforgeeks.org/software-engineering-classification-of-softwarerequirements/?ref=lbp

https://www.geeksforgeeks.org/how-to-write-a-good-srs-for-your-project/?ref=lbp

https://www.geeksforgeeks.org/software-engineering-quality-characteristics-of-a-goodsrs/?ref=lbp

https://www.geeksforgeeks.org/software-engineering-requirements-elicitation/?ref=lbp

https://www.geeksforgeeks.org/software-engineering-challenges-eliciting-requirements/?ref=lbp

Table of Contents

Software Engineering | Requirements Engineering Process

https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/?ref=lbp

- Difficulty Level : Easy
- Last Updated : 27 Feb, 2020

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system. Requirements Engineering Process consists of the following main activities:

- Requirements elicitation
- Requirements specification
- Requirements verification and validation
- Requirements management

Requirements Elicitation:

It is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of same type, standards and other stakeholders of the project.
The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc. Some of these are discussed here. Elicitation does not produce formal models of the requirements understood. Instead, it widens the domain knowledge of the analyst and thus helps in providing input to the next stage.

Requirements specification:

This activity is used to produce formal software requirement models. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process.
The models used at this stage include ER diagrams, data flow diagrams(DFDs), function decomposition diagrams(FDDs), data dictionaries, etc.

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 management:

Requirement management is the process of analyzing, documenting, tracking, prioritizing and agreeing on the

requirement and controlling the communication to relevant stakeholders. This stage takes care of the changing nature of requirements. It should be ensured that the SRS is as modifiable as possible so as to incorporate changes in requirements specified by the end users at later stages too. Being able to modify the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

Software Engineering | Classification of Software Requirements

https://www.geeksforgeeks.org/software-engineering-classification-of-software-requirements/?ref=lbp

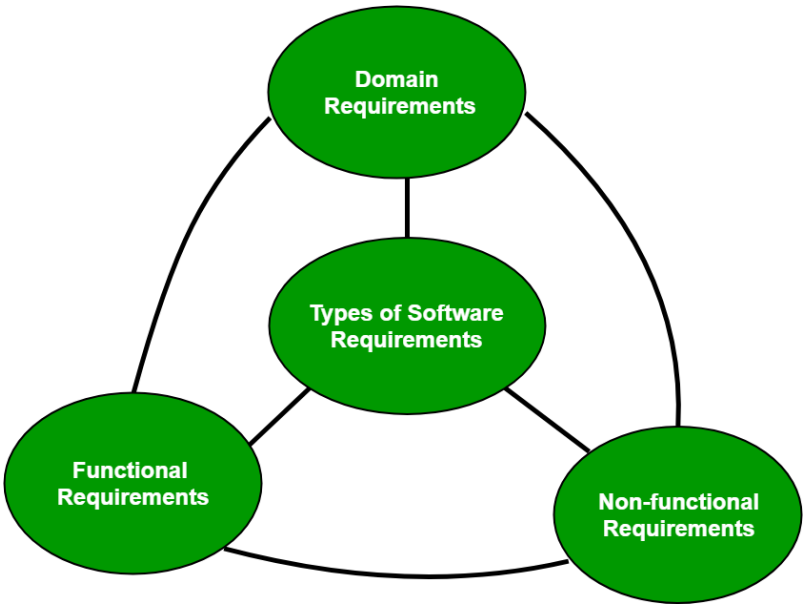
- Difficulty Level : Easy
- Last Updated : 19 Jun, 2018

According to IEEE standard 729, a requirement is defined as follows:

- A condition or capability needed by a user to solve a problem or achieve an objective
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
- A documented representation of a condition or capability as in 1 and 2.

A software requirement can be of 3 types:

- Functional requirements
- Non-functional requirements
- Domain 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.

For example, in a hospital management system, a doctor should be able to retrieve the information of his patients. Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world. In order to accurately describe the functional requirements, all scenarios must be enumerated.

There are many ways of expressing functional requirements e.g., natural language, a structured or formatted language with no rigorous syntax and formal specification language with proper syntax.

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

NFR's are classified into following types:

- Interface constraints
- Performance constraints: response time, security, storage space, etc.
- Operating constraints
- Life cycle constraints: mantainability, portability, etc.
- Economic constraints

The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate.

Domain requirements:

Domain requirements are the requirements which are characteristic of a particular category or domain of projects. The basic functions that a system of a specific domain must necessarily exhibit come under this category. For instance, in an academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement. These requirements are therefore identified from that domain model and are not user specific.

How to write a good SRS for your Project

https://www.geeksforgeeks.org/how-to-write-a-good-srs-for-your-project/?ref=lbp

- Difficulty Level : Easy
- Last Updated : 29 Feb, 2016

What is SRS?

A software requirements specification (SRS) is a description of a software system to be developed. It lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide.

Why SRS?

In order to fully understand one's project, it is very important that they come up with a SRS listing out their requirements, how are they going to meet it and how will they complete the project. It helps the team to save upon their time as they are able to comprehend how are going to go about the project. Doing this also enables the team to find out about the limitations and risks early on.

The following is a sample SRS that I wrote for one of my project.

Project Plan: MeetUrMate

1. Introduction

This document lays out a project plan for the development of "MeetUrMate" open source repository system by Anurag Mishra.

The intended readers of this document are current and future developers working on "MeetUrMate" and the sponsors of the project. The plan will include, but is not restricted to, a summary of the system functionality, the scope of the project from the perspective of the "MeetUrMate" team (me and my mentors), scheduling and delivery estimates, project risks and how those risks will be mitigated, the process by which I will develop the project, and metrics and measurements that will be recorded throughout the project.

2. Overview

In today's world, owning to the heavy workload on the employees, they are having huge amount of stress in their lives. Even with the presence of so many gadgets in and around them, they are not able to relieve their stress. I aim to develop an application that would enable them to share the thing of their liking and meet the person who have the same passion as theirs. For eg. If someone wants to share their art, they can share it through the platform, if someone wants to sing any song, they can record it and share the same. They can also share videos (with some funny commentary in the background), share mysteries which other people can solve, post any question. Through my platform, I'll enable them to meet people who share the common interests and passion, chat with them and have some fun.

2.1 Customers

Everyone. Anyone can use this application ranging from a child to an old-age person.

2.2 Functionality

- Users should be able to register through their already existing accounts.
- They should be able to share snaps/videos/snaps.
- People should be able to like and comment on any post. One person can follow another person who share common interests and likings which would enable them to find mates apart from their usual friend circle.
- Each user can have his/her profile picture, status
- People can post mysteries and other people can solve the mysteries.

- Users will get points for the popularity of their posts/the number of mysteries they solve.
- *Add own funny commentary on any video*
- *Post any questions regarding their interests and people can answer.*

P.S. Italic points features can be inculcated later.

2.3 Platform

It will be launched both as a Web-based application and Mobile app for Android.

2.4 Development Responsibility

I, Anurag Mishra, would be developing the software and I am responsible for the creation of the Database and all the other related stuffs.

3. Goals and Scopes

- Users should be able to register through their already existing accounts.
- They should be able to share snaps/videos/snaps.
- People should be able to like and comment on any post.
- One person can follow another person who share common interests and likings which would enable them to find mates apart from their usual friend circle.
- Each user can have his/her profile picture, status.
- People can post mysteries and other people can solve the mysteries.
- Users will get points for the popularity of their posts/the number of mysteries they solve.

4. Deliverables

I'll deliver the following during the course of development:

- Feature specification
- Product design
- Test plan
- Development document
- Source code

5. Risk Management

5.1 Risk Identification

Following will be the risk involved in my project:

1) People are already using Facebook to find friends. So, what would be the real cause that would motivate them to join my application.

5.2 Risk Mitigation

Even though most of the users would already be using Facebook, our platform would still offer them many things that is not there on Facebook. For eg.

1. They don't meet people who share common interests and passions as much. Our application would enable them to meet people (apart from usual friends) who share common interests and passions on a more frequent basis.
2. Users of fb cannot share songs on-the-go which they have sung whereas on our app they can do that on-the-go.

3. People can post mysteries/cases and other people can solve it. Moreover, people will get points in case they solve the mysteries or on the basis of popularity of their posts.
4. More importantly, people need not register for my application, but instead, they can login using their already existing accounts of Google/Facebook.

Thus, I think that there is a considerable amount of difference between Facebook/Instagram/Twitter and my application and it would attract many people.

6. Scheduling and Estimates

Milestone	**Description**	**Release Date**	**Release Iteration**
M1	Application view and Design (Front-end development)	October 5, 2015	R1
M2	Database for my application (Back-end)	October 17, 2015	R1
M3	Integrating views and designs (Integrating front-end and back-end)	November 12, 2015	R1
M4	Testing for initial release	November 20, 20015	R2
M5	Issue tracker, user reviews, web design integration	December 1, 2015	R2
M6	Final release	December 23, 2015	R2

7. Technical Process

Following would be the languages I would use to develop my application within the stipulated time period:

Front-end development: Jquery, HTML, CSS, PHP.

Back-end development: PHP, MySQL.

For Android app: Java on Android SDK.

The blog is contributed by Anurag Mishra.

Software Engineering | Quality Characteristics of a good SRS

https://www.geeksforgeeks.org/software-engineering-quality-characteristics-of-a-good-srs/?ref=lbp

- Difficulty Level : Easy
- Last Updated : 05 Oct, 2020

Related Article: Writing a good SRS for your project

Following are the characteristics of a good SRS document:

1. ## Correctness:
   User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.

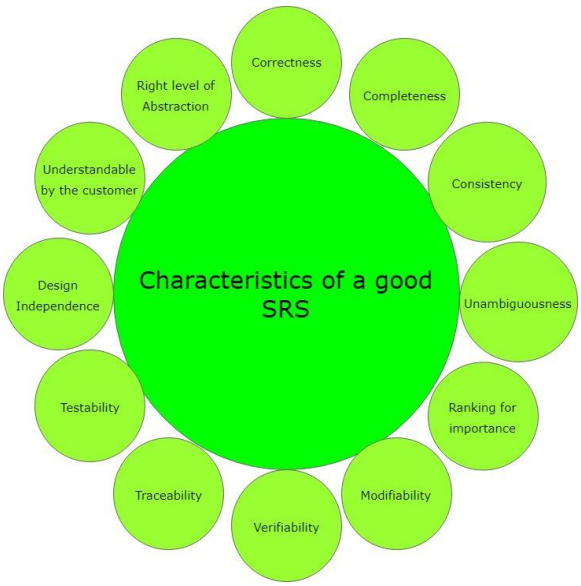
2. ## Completeness:
   Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.

3. ## Consistency:
   Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

4. ## Unambiguousness:
   A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation. Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

5. ## Ranking for importance and stability:
   There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.

6. ## Modifiability:
   SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent. Modifications should be properly indexed and cross-referenced.



7. ### Verifiability:
   A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement stating that the system must be user-friendly is not verifiable and listing such requirements should be avoided.

8. ### Traceability:
   One should be able to trace a requirement to design component and then to code segment in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.

9. ### Design Independence:
   There should be an option to choose from multiple design alternatives for the final system. More specifically, the SRS should not include any implementation details.

10. ### Testability:
    A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

11. ### Understandable by the customer:
    An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.

12. ### Right level of abstraction:
    If the SRS is written for the requirements phase, the details should be explained explicitly. Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

Software Engineering | Requirements Elicitation

https://www.geeksforgeeks.org/software-engineering-requirements-elicitation/?ref=lbp

- Difficulty Level : Basic
- Last Updated : 22 Mar, 2021

Requirements elicitation is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need.

Requirements elicitation Activities:

Requirements elicitation includes the subsequent activities. Few of them are listed below –

- Knowledge of the overall area where the systems is applied.
- The details of the precise customer problem where the system are going to be applied must be understood.
- Interaction of system with external requirements.
- Detailed investigation of user needs.
- Define the constraints for system development.

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.

Software Engineering | Challenges in eliciting requirements

https://www.geeksforgeeks.org/software-engineering-challenges-eliciting-requirements/?ref=lbp

- Last Updated : 22 Mar, 2021

Prerequisite – Requirements Elicitation

Eliciting requirements is the first step of Requirement Engineering process. It helps the analyst to gain knowledge about the problem domain which in turn is used to produce a formal specification of the software. There are a number of issues and challenges encountered during this process. Some of them are as follows:

1. **Understanding large and complex system requirements is difficult –**
   The word 'large' represents 2 aspects:
   - (i) Large constraints in terms of security, etc. due to a large number of users.
   - (ii) Large number of functions to be implemented.
2. **Undefined system boundaries –**
   There might be no defined set of implementation requirements. The customer may go on to include several unrelated and unnecessary functions besides the important ones, resulting in an extremely large implementation cost which may exceed the decided budget.
3. **Customers/Stakeholders are not clear about their needs. –**
   Sometimes, the customers themselves maybe unsure about the exhaustive list of functionalities they wish to see in the software. This might happen when they have a very basic idea about their needs but haven't planned much about the implementation part.
4. **Conflicting requirements are there –**
   There is a possibility that two different stakeholders of the project express demands which contradict each other's implementation. Also, a single stakeholder might also sometimes express two incompatible requirements.
5. **Changing requirements is another issue –**
   In case of successive interviews or reviews from the customer, there is a possibility that the customer expresses a change in the initial set of specified requirements. While it is easy to accommodate some of the requirements, it is often difficult to deal with such changing requirements.
6. **Partitioning the system suitably to reduce complexity –**
   The projects can sometimes be broken down into small modules or functionalities which are then handled by separate teams. Often, more complex and large projects require more partitioning. It needs to be ensured that the partitions are non-overlapping and independent of each other.
7. **Validating and Tracing requirements –**
   Cross-checking the listed requirements before starting the implementation part is very important. Also, there should be forward as well as backward traceability. For eg, all the entity names should be the same everywhere, i.e., there shouldn't be a case where 'STUDENT' and 'STUDENTS' are used at separate places to refer to the same entity.
8. **Identifying critical requirements –**
   Identifying the set of requirements which have to be implemented at any cost is very important. The requirements should be prioritized so that crucial ones can be implemented first with the highest priority.

9. Resolving the “to be determined” part of the requirements –
   The TBD set of requirements include those requirements which are yet to be resolved in the future. The number of such requirements should be kept as low as possible.

10. Proper documentation, proper meeting time and budget constraints –
    Ensuring a proper documentation is an inherent challenge, especially in case of changing requirements. The time and budget constraints too need to be handled carefully and systematically.

Summary

There are many problems associated with the engineering of needs, including problems in defining the scope of the system, problems with promoting understanding between the various communities affected by the proposed system, and problems in addressing the instability of demand. These problems can create unsustainable needs and cancel program development, otherwise, the development of a system will be considered unsatisfactory or unacceptable, have high repair costs, or will be subject to change. By improving service delivery, the needs engineering process can be improved, which has resulted in improved system requirements and a better system.

Engineering needs can be deduced from the requirements of promotion, specification, and validation requirements. Most current strategies and needs focus on clarity, that is, service representation. The report focuses on rather than concerns about nominations, those issues with engineering needs that have not been adequately addressed by specification strategies. A lifting approach is proposed to address these concerns.

This new approach seeks to incorporate the benefits of existing promotion strategies while taking a closer look at the activities undertaken during the service delivery process. These activities include fact-finding, data collection, evaluation and planning, prioritization, and integration. Taken on their own, the existing promotion strategies are lacking in one or more of these areas