Software Development 2019

FACT SHEET – SRS

What is a Software Requirements Specification (SRS)?

Software requirements specification is a comprehensive description of the intended purpose and environment for purpose-designed software solutions. It documents the key activities associated with the analysing stage of the problem-solving methodology. Software requirements specifications (SRS) fulfil the purposes of breaking down a problem into component parts, providing input to the design stage and serving as a reference point for further stages of the problem-solving methodology.

VCE Computing Study Design 2016–2020 pg. 13

Software Requirements Specification (SRS): A comprehensive description of the intended purpose and environment for purpose-designed software solutions. It documents the key tasks associated with the **analysing** stage of the **problem solving methodology**. The software requirements specifications (SRS) fully describes the functional requirements (what it is required to do) and non-functional requirements of the solution (solution attributes) such as user-friendliness, response rates, robustness, portability, reliability and maintainability, the conditions affecting the solution (constraints) and the parameters of the solution (scope). Software requirements specifications fulfils the purposes of breaking down the problem into component parts, providing input to the designing stage and serving as a reference point for further stages of the problem-solving methodology.

VCE Information Technology Study Design 2011–2014 pg. 14 – please note old study definition, but provided additional depth in definition.

See Sample SRS Template – provided.

It's important to note that an SRS contains functional and nonfunctional requirements only; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

A well-designed, well-written SRS accomplishes four major goals:

- It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, the SRS should be written in natural language (versus a formal language, explained later in this article), in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.
- It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.
- It serves as an input to the design specification. As mentioned previously, the SRS serves as the parent document to subsequent documents, such as the software design specification and statement of work. Therefore, the SRS must contain sufficient detail in the functional system requirements so that a design solution can be devised.
- It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

http://www.techwr-l.com/techwhirl/magazine/writing/softwarerequirementspecs.html, <21/03/11, checked 4/3/19>

The IEEE (www.ieee.org) is an excellent source for definitions of System and Software Specifications.
In its standards document (IEEE830) it states: To the customers, suppliers, and other individuals, a good SRS should provide several specific benefits such as the following:

- *Establish the basis for agreement between the customers and the suppliers on what the software product is to do.*
- *Reduce the development effort.*
- *Provide a basis for estimating costs and schedules.*
- *Provide a baseline for validation and verification*
- *Facilitate transfer.(implementation in other users)*
- *Serve as a basis for enhancement*

It suggests that an SRS should be

a) Correct
b) Unambiguous
c) Complete
d) Consistent
e) Ranked for importance and/or stability
f) Verifiable
g) Modifiable
h) Traceable

Correct - Of course you want the specification to be correct. No one writes a specification that they know is incorrect. We like to say - "Correct and Ever Correcting." The discipline is keeping the specification up to date when you find things that are not correct.
Unambiguous - An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. **Complete -** A simple judge of this is that is should be all that is needed by the software designers to create the software.
Consistent - The SRS should be consistent within itself and consistent to its reference documents.
Ranked for Importance - Very often a new system has requirements that are really marketing wish lists. Some may not be achievable. It is useful provide this information in the SRS.
Verifiable - Don't put in requirements like - "It should provide the user a fast response." Or "The system should never crash." Instead, provide a quantitative requirement like: "Every key stroke should provide a user response within 100 milliseconds."
Modifiable - Having the same requirement in more than one place may not be wrong - but tends to make the document not maintainable.
Traceable - Often, this is not important in a non-politicized environment. However, in most organizations, it is sometimes useful to connect the requirements in the SRS to a higher level document. Why do we need this requirement?

http://www.microtoolsinc.com/Howsrs.php, <21/03/11, last checked 4/3/19>

What is involved in the analysis stage of the problem solving methodology (PSM)?

Analysis typically answers the **'what questions'** – what is needed to solve a problem, given particular circumstances?
It involves:

- **Determining the solution requirements**. What output is the solution to provide? What data is needed to produce the output? What functions must the solution provide? These requirements can be classified as being **functional**, that is, what the solution is required to do, and **non-functional**, which describes the **attributes** the solution should possess including **useability, reliability, portability, robustness, maintainability**. Tools to assist in determining the solution requirements include context diagrams, data flow diagrams and use cases.
- **Identifying the constraints on the solution**. What conditions need to be considered when designing a solution? Typical constraints include **economic**, such as cost and time; **technical**, such as speed of processing, capacity, availability of equipment, compatibility and security; **social**, such as level of expertise of users; **legal**, such as ownership and privacy of data requirements; and **useability**, such as usefulness and ease of use of solutions.
- **Determining the scope of the solution**. The scope states the **boundaries** or parameters of the solution. It identifies the area of interest or what aspects of the problem will and will not be addressed by the solution.

THE SOFTWARE REQUIREMENT SPECIFICATION TEMPLATE

What is the purpose?

The purpose (or product vision) describes what the software product is about and what it eventually could become.
The classic way to validate the product vision is to answer the elevator test: "Can you explain your product in the time it takes to ride up in an elevator?" Moore (2006, p. 152). Passing this test ensures that your product vision is clear, engaging, and brief (assuming we ride up a building with the right height and don't get stuck).
https://www.scrumalliance.org/community/articles/2009/january/the-product-vision, <last accessed: 4/3/19>

The following keyword template from Geoffrey Moore's book Crossing the Chasm, provides a framework for developing the purpose used in the SRS

- For (target customer)
- Who (statement of the need or opportunity)
- The (product name) is a (product category)
- That (key benefit, compelling reason to use)
- Unlike (current system)
- This product (advantages of new product)

Example – Purpose or product vision statement for a Chemical Tracking System

***For** scientists **who** need to request containers of chemicals, **the** Chemical Tracking System **is** an information system **that** will provide a single point of access to the chemical stockroom and to vendors. [The system will store the location of every chemical container within the company, the quantity of material remaining in it, and the compete history of each container's location and usage. This system will save the company 25 per cent on chemical costs in the first year of use by allowing the company to fully exploit chemicals that are already available within the company, dispose of fewer partially use or expired containers, and use a single standard chemical purchasing process.] **Unlike** the current manual ordering processes, **our product** will generate all reports required to comply with federal and state government regulations that require the reporting of chemical usage, storage and disposal. Wiegers, K.E.; Software Requirements, Microsoft Press, 2003, pg. 85-86*

*Note: The smaller red font is a full complete purpose statement; it is not a **must** in this course of study.*

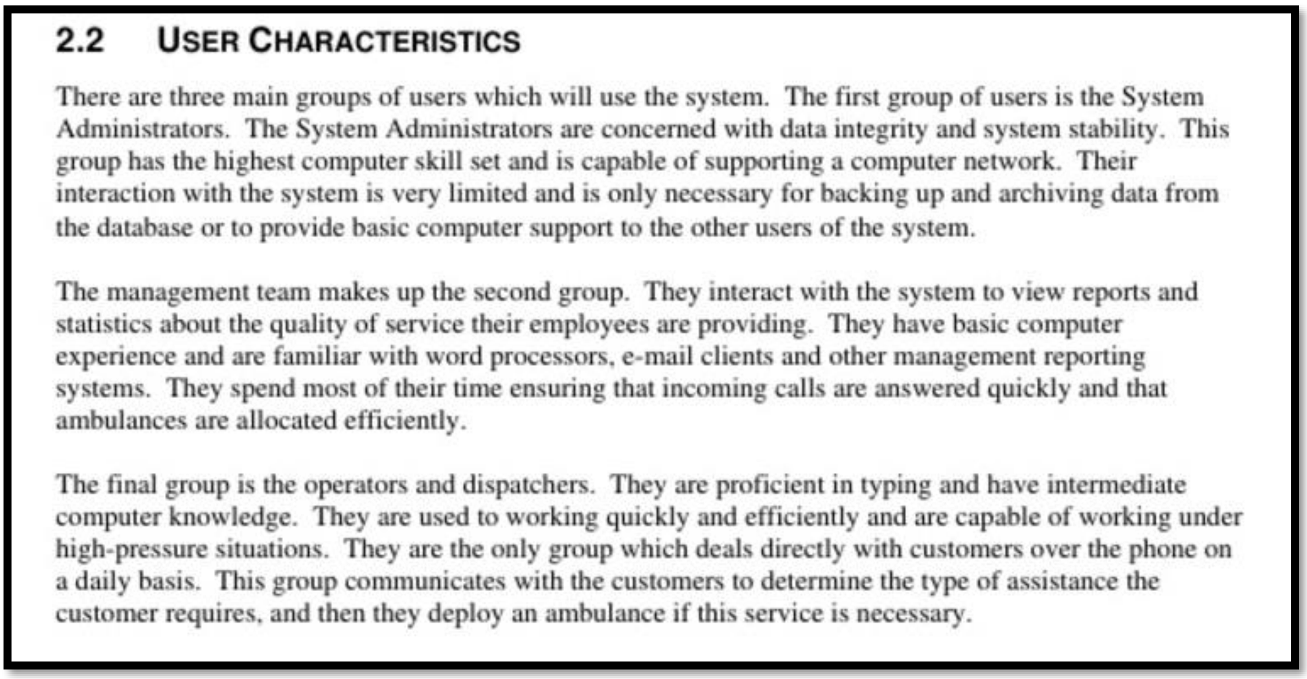
What are the user characteristics?

This subsection of the SRS should describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. It should not be used to state specific requirements, but rather should provide the reasons why certain specific requirements are later specified in the SRS.
http://www.cse.msu.edu/~cse870/IEEEXplore-SRS-template.pdf <4/3/19>

You may also find the section on "finding the voice of the user' helpful.

https://ptgmedia.pearsoncmg.com/images/9780735679665/samplepages/9780735679665.pdf (last accessed: 4/3/19)

Eg: http://www.slideshare.net/indrisrozas/example-requirements-specification



2.2 User Characteristics

There are three main groups of users which will use the system. The first group of users is the System Administrators. The System Administrators are concerned with data integrity and system stability. This group has the highest computer skill set and is capable of supporting a computer network. Their interaction with the system is very limited and is only necessary for backing up and archiving data from the database or to provide basic computer support to the other users of the system.

The management team makes up the second group. They interact with the system to view reports and statistics about the quality of service their employees are providing. They have basic computer experience and are familiar with word processors, e-mail clients and other management reporting systems. They spend most of their time ensuring that incoming calls are answered quickly and that ambulances are allocated efficiently.

The final group is the operators and dispatchers. They are proficient in typing and have intermediate computer knowledge. They are used to working quickly and efficiently and are capable of working under high-pressure situations. They are the only group which deals directly with customers over the phone on a daily basis. This group communicates with the customers to determine the type of assistance the customer requires, and then they deploy an ambulance if this service is necessary.

What is the Scope?

Wieger's defines scope as the portion of the ultimate product vision that the current project will address. The scope draws the boundary between what's in and what's out for the project.

Stellman & Greene define the scope as

This section contains a brief description of the scope of the document. If the SRS is a complete description of the software, then it will state something similar to: "This document contains a complete description of the functionality of the (name of project) project. It consists of use cases, functional requirements and non functional requirements, which, taken together form a complete description of the software." For complex software, the requirements for the project might be divided into several SRS documents. In this case, the scope should indicate which portion of the project is covered in this document.
http://www.stellman-greene.com/images/stories/Library/SRS%20Outline.pdf, <20/03/11> (Archived copy can be found at https://web.archive.org/web/20110812171648/http://www.stellman-greene.com/images/stories/LectureNotes/06%20requirements.pdf accessed 4/3/19)

What is the Operating Environment?

Wieger suggests that this section of the SRS should describe the environment in which the software will operate, including the hardware platform, the operating systems and versions, and the geographical locations of users, servers and databases. It should list any other software components or applications with which the system must peacefully coexist.

Wiegers, K.E.; Software Requirements, Microsoft Press, 2003, pg. 174

Application Architecture

In your SRS, you have to discuss the application architecture. There are *styles of modern application architecture, including mobile, rich client, peer-to-peer and internet applications*

- Thick vs. Thin Client link
- Client-Server link
- P2P link
- Rich Internet architecture link

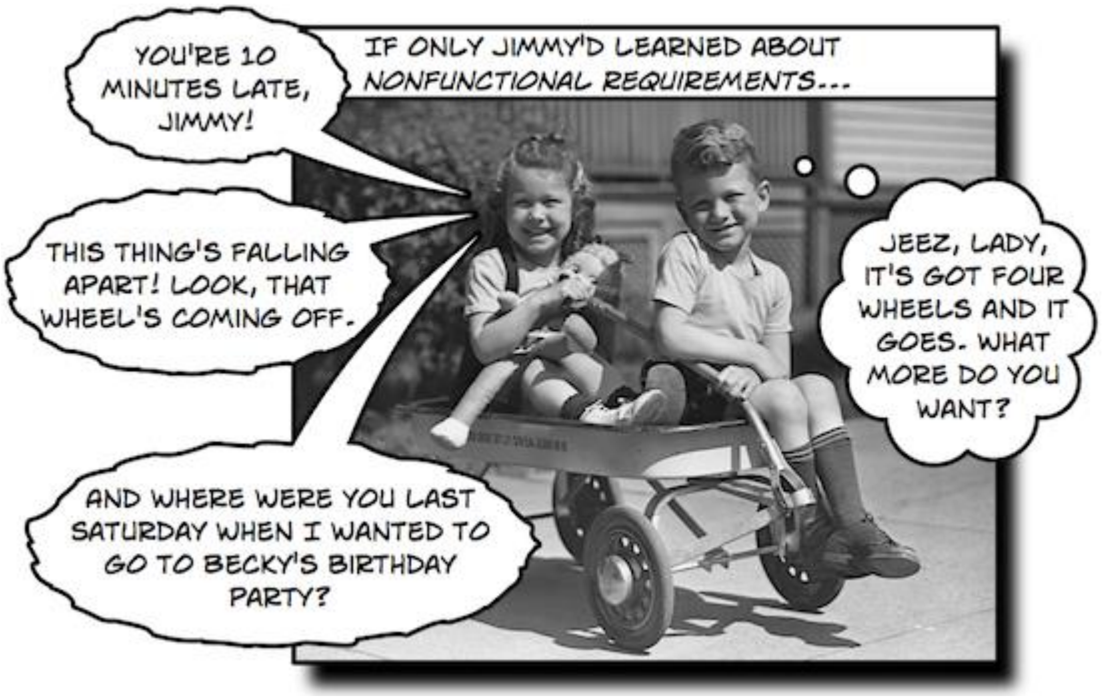
What are Functional Requirements?

Wikipedia provided the following definition:

> *In software engineering, a **functional requirement** defines a function of a software system or its component. A function is described as a set of inputs, the behavior, and outputs (see also software). Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases.*

http://en.wikipedia.org/wiki/Functional_requirements. <21/03/11, checked 9 April 2016>

What are Non Functional Requirements?


YOU'RE 10 MINUTES LATE, JIMMY!
IF ONLY JIMMY'D LEARNED ABOUT NONFUNCTIONAL REQUIREMENTS...
THIS THING'S FALLING APART! LOOK, THAT WHEEL'S COMING OFF.
JEEZ, LADY, IT'S GOT FOUR WHEELS AND IT GOES. WHAT MORE DO YOU WANT?
AND WHERE WERE YOU LAST SATURDAY WHEN I WANTED TO GO TO BECKY'S BIRTHDAY PARTY?

Using non-functional requirements to build better software, <http://www.stellman-greene.com/2009/10/03/using-nonfunctional-requirements/>, 14/03/11, check 9 April 2016>

Stellman comments in *Using non-functional requirements* to build better software,   'understanding non-functional requirements — which some people call software quality attributes or non behavioural requirements — can make a big difference when you're building software'.

Users have implicit expectations about how well the software will work. These characteristics include how easy the software is to use, how quickly it executes, how reliable it is, and how well it behaves when unexpected conditions arise. The non-functional requirements define these aspects about the system.

The non-functional requirements should be defined as precisely as possible. Often, this is done by quantifying them. Where possible, the non-functional requirements should provide specific measurements that the software must meet. The maximum number of seconds it must take to perform a task, the maximum size of a database on disk, the number of hours per day a system must be available, and the number of concurrent users supported are examples of requirements that the software must implement but do not change its behaviour.

Wiegers in *Software Requirements* defines a quality attribute as 'a kind of non-functional requirement that describes a quality or property of the system. Examples include usability, portability, maintainability, integrity, efficiency, reliability and robustness. Quality attribute requirements describe the extent to which a software product demonstrates desired characteristics, not what the product does.'

Stellman refers to the following requirements checklist,

- **Availability:** A system's availability, or "uptime," is the amount of time that it is operational and available for use. This is specified because some systems are designed with expected downtime for activities like database upgrades and backups.
- **Efficiency:** Specifies how well the software utilizes scarce resources: CPU cycles, disk space, memory, bandwidth, etc.
- **Flexibility:** If the organization intends to increase or extend the functionality of the software after it is deployed, that should be planned from the beginning; it influences choices made during the design, development, testing, and deployment of the system.
- **Portability:** Portability specifies the ease with which the software can be installed on all necessary platforms, and the platforms on which it is expected to run.
- **Integrity:** Integrity requirements define the security attributes of the system, restricting access to features or data to certain users and protecting the privacy of data entered into the software.
- **Performance:** The performance constraints specify the timing characteristics of the software. Certain tasks or features are more time-sensitive than others; the non-functional requirements should identify those software functions that have constraints on their performance.
- **Reliability:** Reliability specifies the capability of the software to maintain its performance over time. Unreliable software fails frequently, and certain tasks are more sensitive to failure (for example, because they cannot be restarted, or because they must be run at a certain time).
- **Reusability:** Many systems are developed with the ability to leverage common components across multiple products. Reusability indicates the extent to which software components should be designed in such a way that they can be used in applications other than the ones for which they were initially developed.
- **Robustness:** A robust system is able to handle error conditions gracefully, without failure. This includes a tolerance of invalid data, software defects, and unexpected operating conditions.
- **Scalability:** Software that is scalable has the ability to handle a wide variety of system configuration sizes. The non-functional requirements should specify the ways in which the system may be expected to scale up (by increasing hardware capacity, adding machines, etc.).
- **Usability:** Ease-of-use requirements address the factors that constitute the capacity of the software to be understood, learned, and used by its intended users.

Andrew Stellman & Jennifer Greene, Applied Software Project Management, chapter 6, page 113 (O'Reilly 2005)

The ***VCE Computing Study Design 2016–2020 pg. 13*** refers to the follow **non-functional requirements** including usability, reliability, portability, robustness, maintainability. These are the terms on which you can be examined; ensure you have a good working description of each term.

What is a Constraint?

A constraint is a restriction that is imposed on the choices available to the developer for the design and construction of a product. Typical constraints include cost, speed of processing, requirements of users, legal requirements, security, compatibility, level of expertise, capacity, availability of equipment, etc.

Constraints generally fall under the following headings – performance, design and implementation and hardware.

Context Diagrams & Dataflow Diagrams

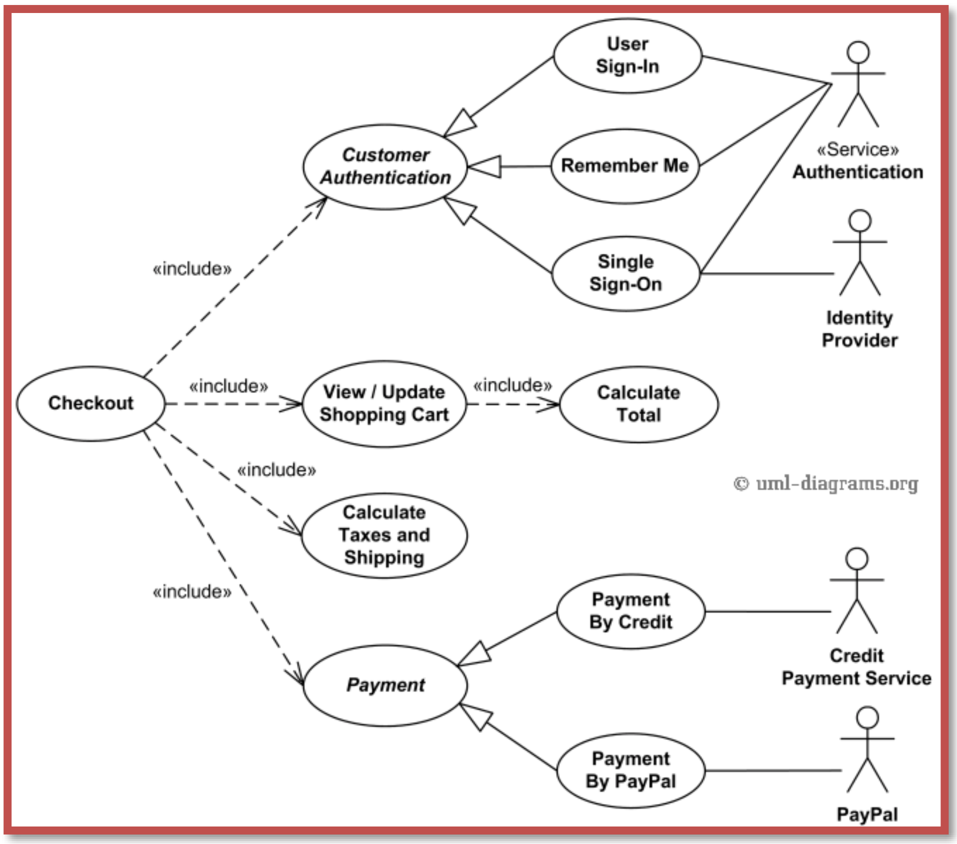
See text: Adams, J.; Systems Documentation Methods, Eastern House, 1997

Use Case and Use Case Diagrams

This section contains a set of use cases that describe the external behaviour of the software. A use case is a description of a specific interaction that a user may have with the software. Use cases are deceptively simple tools for describing the functionality of the software. A use case is a simple, straightforward tool that can be used to completely describe all of the behaviour of a piece of software. It contains a textual description of all of the ways that the intended users could work with the software through its interface. Use cases do not describe any internal workings of the software, nor do they explain how that software will be implemented. They simply show how the steps that the user follows to use the software to do his work. All of the ways that the users interact with the software can be described in this manner.

http://www.stellman-greene.com/images/stories/Library/SRS%20Outline.pdf, <20/03/11, unavailable 9 April 2016, a version can be found at http://www.stellman-greene.com/about/applied-software-project-management/applied-software-project-management-product-engineering-practices/#Software_Requirements_Specification/ >

http://www.uml-diagrams.org/examples/online-shopping-use-case-diagram-example.html


User Sign-In
Customer Authentication
Remember Me
«Service» Authentication
«include»
Single Sign-On
Identity Provider
Checkout
«include»
View / Update Shopping Cart
«include»
Calculate Total
«include»
© uml-diagrams.org
Calculate Taxes and Shipping
«include»
Payment By Credit
Credit Payment Service
Payment
Payment By PayPal
PayPal