Chapter 2
Understanding Requirements

We have defined the term "requirement" in Chap. 1 as applied in the context of requirements management in software development. Now let us discuss the requirements in greater detail so that we understand the term in its entirety.

2.1 Classification of Requirements

Requirements can be classified based on three considerations, namely,

1. Functionality considerations—these are the requirements that fulfill the set of selected business processes and deliver the results to end-users.
2. Product construction considerations—these are the requirements that are necessary to build the product efficiently as well as to maintain it later on.
3. Source considerations—requirements for software development are provided from different sources. This classification is based on the agencies that provide the requirements.

2.2 Classification of Requirements Based on Functionality Considerations

Requirements can be classified into two major classes from the functionality standpoint, namely,

1. Core functionality requirements
2. Ancillary functionality requirements

M. Chemuturi, *Requirements Engineering and Management for Software Development Projects*, DOI: 10.1007/978-1-4614-5377-2_2,

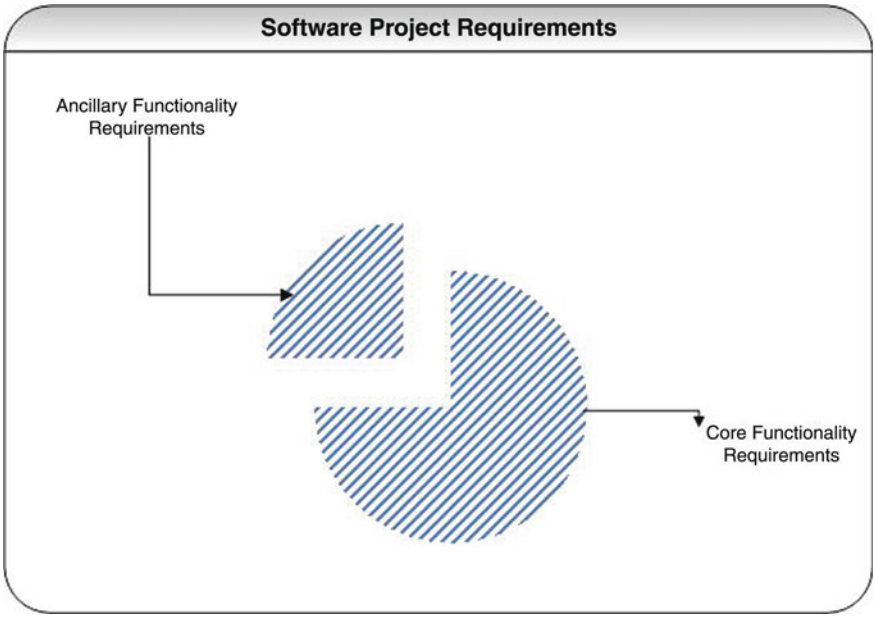

Fig. 2.1 Classification of software project requirements based on functionality

Core functionality requirements are those functionalities of the product without which, the product is not useful for the users. These functionalities must be fulfilled and exceptions cannot be granted or resorted to during development of the product. Core functionality addresses the performance of a set of business processes. The main purpose of software development is to fulfill this core functionality.

Ancillary functionality requirements supplement core functionality. Even if the ancillary functionality is not fulfilled, the product is still useful but may cause inconvenience in the form of loss of productivity or security. One significant point to be noted here is that the customer, more often than not, may not specify this functionality and even expects the development team to take care of this ancillary functionality!

Figure 2.1 depicts the classification based on functionality pictorially.

Classification of requirements based on functionality consideration is sometimes classified as (a) Functional Requirements and (b) Non-Functional Requirements. When we prefix the word "non" to any other word, it connotes the opposite of the word. When we say "non-functional" it has connotation that the requirements do not function or do not serve any function. In reality these requirements may not serve business process functions directly but they are serving a useful purpose in the software, They indirectly, perhaps, assist in the better functioning of business processes. User-friendliness does not serve any business process but if we take it away from our software, using the software and performing the business

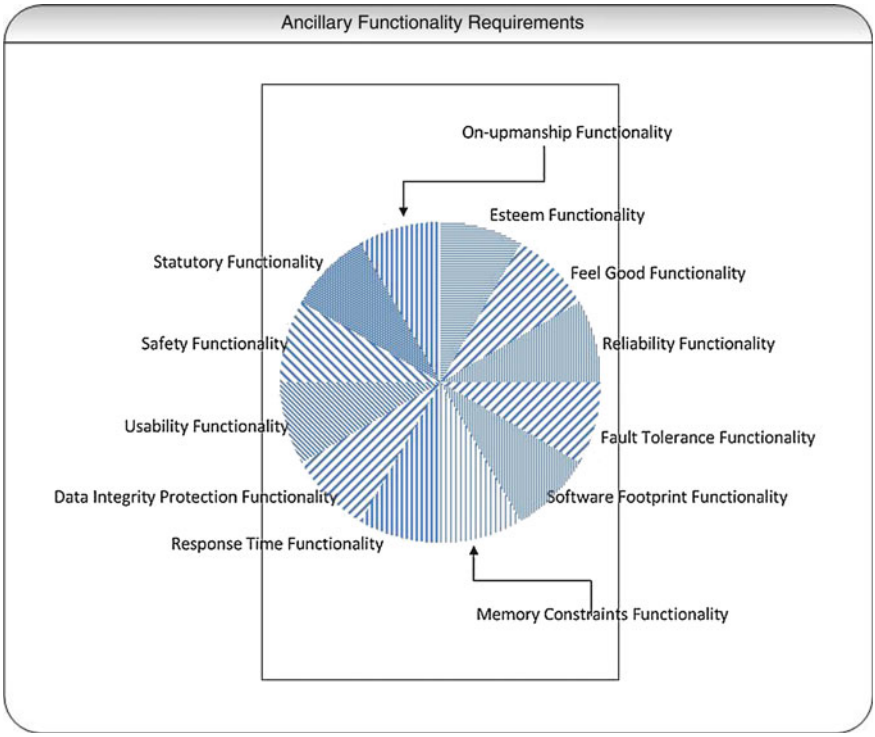

Fig. 2.2 Ancillary functionality requirements

processes using the software becomes tedious. That is the reason, I have not used the "functional and non-functional" classification of requirements.

In almost all cases of software development, core functionality is addressed completely. The customer provides core functionality in project scenario. It is defined by the project team or domain experts in the case of a product development scenario. But when it comes to ancillary functionality, it is common to miss out on some of the ancillary functionality. In many cases where the end product of software development is panned by the critics, the criticism stems from the fact that all relevant ancillary functionality is not fully or not efficiently implemented.

Various classes of statutory functionality requirements is depicted pictorially in Fig. 2.2.

The following are the ancillary functionalities that need to be included in the overall requirements for the project:

1. **Statutory functionality**—This functionality is required because of regulations/standards of government, or industry associations or professional associations. In software development industry we have Institute of Electrical and Electronics Engineers (IEEE), Software Process Improvement Networks (SPINs), Software Project Managers Networks (SPMNs) in addition to governments.

The functionality to enable persons with disabilities to use the product as also the functionality to ensure that intruder prevention come under this category.

2. **Safety functionality**—This functionality protects the user from causing injury to the users. However, as computer usage especially from this stand point cannot cause physical injury, it has to protect him from logical injuries including financial losses. Examples for safety functionality include charging the credit card but not booking a ticket in online transactions, computational inaccuracies, performing wrong transaction (buying shares when we intended to sell), and data loss due to accidental deletes.
3. **Security functionality**—While safety functionality is to protect the user from the product, security is safety from external attacks. Protection from intruders, malicious use of insiders, and data theft via the Internet and so on are examples of security functionality.
4. **Usability functionality**—It was originally referred to as User-friendliness. The objective of this functionality is to make the software usable intuitively and with minimal reference to user manuals. Graphical user Interface (GUI) has achieved much of this functionality but improvement is always possible.
5. **Data Integrity Protection functionality**—Now that computing shifted from EDP (Electronic Data Processing) rooms to end users, this has become an important functionality for any software product. End user can unintentionally and innocently affect data integrity. They may enter numerals in name fields and alphabets in numeric fields. In Human Resources (HR) applications, they may enter a date of birth such that the age of the employee may either be below legal employment age or above the retirement age. In hospital or hotel management applications, they may enter check out date as prior to check in date. Users can enter wrong data in a hoard of ways. It is essential that all necessary actions must be taken to prevent entry of wrong data. So data validation requirements become important aspect of ancillary functionality.
6. **Response time functionality**—Sometimes, especially in real time applications, response times form part of core functionality. In business applications however, they form part of ancillary functionality. In web based applications, response times are important as the user and the server may not be at one location and if the application does not respond fast enough, the user may abort the application or do something else. Normally these form part of organizational development standards.
7. **Memory constraints functionality**—With software controlling every device today, the constraint still remains. It is a thing of the past as far as computers are concerned but devices like mobile phones, cars, washing machines and the like, not to mention rockets and space shuttles, memory constraints remains. In these cases, memory constraint has to be taken into consideration. These requirements can be obtained from hardware manufacturers who supply the hardware on which the software needs to function.
8. **Software footprint constraint**—When the software resides on a chip in small handheld devices and in various machines, the final size of the package that gets installed becomes very important. Now all the Computer Numerically

Controlled (CNC) machines not only in workshops but also in homes do have this limitation. In the present era, the cars, the refrigerators, the washing machines, the ovens, the mobile phones all have software on a chip which have limited capacity. This calls for restrictions on the size of the software that can be installed on the chip. This information can be obtained from the hardware manufacturers supplying the selected chips on which the software is going to be installed.

9. **Fault tolerance functionality**—Users make mistakes in using software, mostly unintentionally. This functionality ensures that software does not crash or abort when a mistake is committed by the user. It provides an error message and provides an alternative path and allows the user to use other functionality besides coming out of the fault-scenario smoothly without causing any damage.
10. **Reliability functionality**—There is a misconception in some quarters that since there are no moving parts, software ought to function reliably if it correctly runs once. But in these days of fast obsolescence, this becomes applicable. Every 3 years a new version of the operating system, browser, and middleware are released. The hardware, including processors, Network Interface Units (NIUs), switches, etc. are also upgraded every now and then. Since most modern applications work on the Internet the threat of viruses, spyware and malware also increased significantly. Any of these can impact the way our software works. Therefore, it is important to build the software in such a way that some of the possible changes do not affect the software or provide functionality to indicate that the environment changed so that corrective action could be taken. The users should not be faced with a software failure to understand that it needs upgrade. Software failure due to environmental change needs to be trapped and an alternative path needs to be available for the users for smooth changeover or closure of the application.
11. **Feel-good functionality**—This functionality is making the user interface look jazzy and sexy. The idea is to make the users feel good while using the software. This is adding glitz to the user interface screens and using nice pictures for buttons and icons and so on. I know of software product failure because the screens are not sexy!
12. **Esteem functionality**—This functionality brings pride to the users. For example, a Rolex watch delivers the same core functionality (showing time) as any other watch. But by ensuring that the thicker gold plating of the casing, scratch-proof crystal and so on, Rolex watches bring pride to the owners albeit at a higher cost. In software too this kind of functionality can be brought in the form of better messages, handling of error conditions, adding functionality that no other software possesses and so on.
13. **One-upmanship (competitive edge) functionality**—This functionality gives competitive edge to the software. It is having more functionality than any of the competitive software packages possesses. This functionality may be in core functionality or ancillary functionality. Compared feature to feature, this package would have one or more functional aspects in every feature over the competitive software packages.

2.3 Classification of Requirements Based on Product Construction Considerations

Any product, either for use by one organization or multiple organizations, needs to be built such that it works reliably without defects over the life of the product. In the case of software product, the life of the product is not dependent on its structure or wear and tear caused by functioning continuously but is dependent on the hardware platform it is installed on. A software product works as long as its hardware platform is unchanged, in working condition and is maintained well. In these days of fast obsolescence of hardware, the life of computer hardware itself is short compared to other types of machinery. As most of the present day software products are working on the Internet, there are many external factors that can adversely affect the application's reliability and defect-free functioning of the software product. Therefore, utmost care needs to be exercised in constructing the software product so that minor modifications in the environment outside the software and hardware platforms would not impact the reliability and defect-free functioning of the product. These requirements are unlikely to come from the end users or managements of the client organization. These need to be derived and implemented in the software product by the project team and the management of the software development organization. From this standpoint, the classification of requirements is enumerated below. Figure 2.3 depicts these requirements pictorially

1. **Maintainability**—The resultant product ought to be maintainable. That is, it ought to be possible to modify the code, add code or delete some code. The key aspect of maintainability is not just to add, modify or delete but to do those activities efficiently, effectively and with minimum expenditure of resources. Another important aspect of maintainability is that persons other than the original developers should be able to maintain the product. The product specifications that dictate product maintainability come under this category. These are normally in the standards and guidelines selected for the product.
2. **Flexibility**—Flexibility refers to the ability of the product to be used in multiple similar scenarios. For example, a materials management software product should be useful in engineering industry, chemical industry, mass production system and batch production system. Another understanding of flexibility refers to the ability of the product to be useful without any modification, when some of the underlying parameters change. For example, in a payroll software product, addition of a new deduction or addition of a new payment, should not render the product useless. On the other hand, the package should still be usable without changing the source code. The functionality specification that is focused on achieving flexibility in product functioning comes under this category. These specifications are usually part of standards, guidelines and ancillary functionality requirements.
3. **Efficiency**—The resultant product ought to use resources efficiently. The resources used by the product are not just computer resources (CPU, RAM, disk

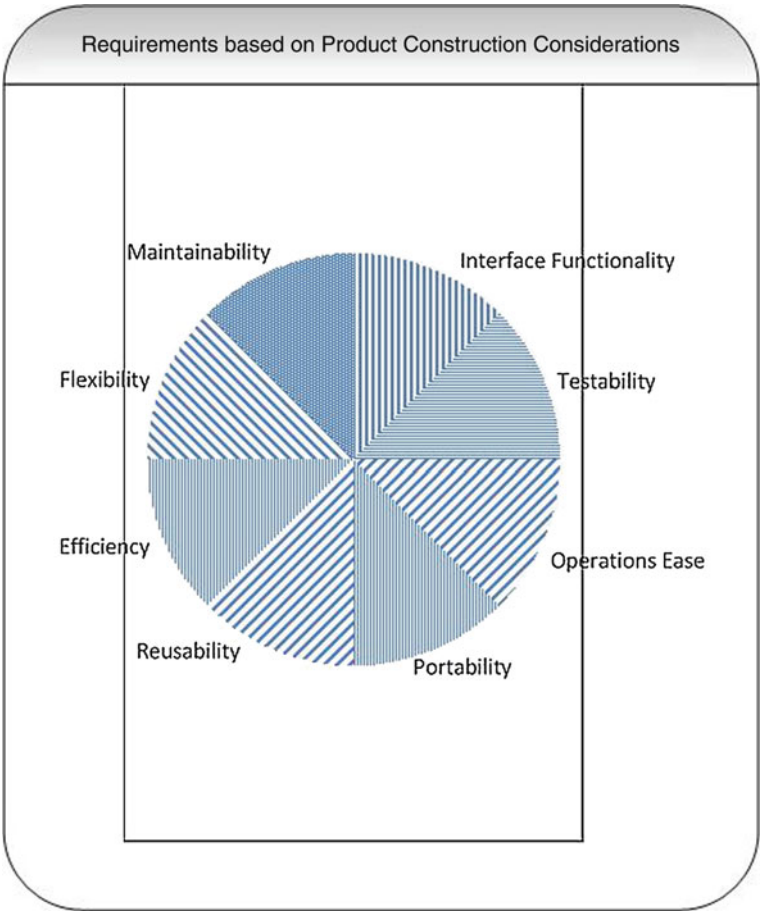

Fig. 2.3 Product construction consideration requirements

space etc.) but also the time of end users, bandwidth on the network, backup storage and so on. The product should minimize the time expected to be spent by the users, perhaps, by reducing the needed key strokes and mouse clicks. The functionality specification that is focused on efficiency of resource usage comes under this category. Standards, guidelines, ancillary functionality requirements provide these requirements.

4. **Reusability**—The product ought to be built is such a way that its components can be used in other products. Automobile industry implements this concept diligently. The same engine is used in multiple models. Components like a steering wheel, brakes, tires, shafts etc. are used without any change in many models. The dictum "There is no point in re-inventing the wheel" seems to

have originated there. In software industry, re-usability is rather an exception rather than a rule. But it pays to implement this concept in software products too. These are non-functional requirements and are normally covered by standards and guidelines or the organization.

5. **Portability**—In earlier days, porting is referred to as shifting the software developed in the same language such as COBOL from one hardware platform to another. But now, such a thing is passé. But another type of porting has come on to the scene. It is shifting the web site from one host to another. With cloud computing, it would be much more frequent in the future to be shifting applications from one host/data center to another. The functionality specifications focused on ensuring that the impact of porting is minimized come under this category and include standards and guidelines for achieving the portability.
6. **Operations ease**—Modern software products are large and multi-functional systems involving many users, perhaps, from different geographical regions spanning international borders. To keep such systems operational round the clock, they need specialists running the operations. Therefore, many of them need dedicated/shared systems administrators, DBAs and network administrators. In many cases, updating the software or hardware has to be achieved without bringing down the systems. Therefore, the software product needs to be built keeping all these aspects in consideration. These aspects are covered normally by the standards and guidelines dealing with product architecture, design and construction.
7. **Testability**—Of course, the product has to be testable and will be so. What is so special about testability then? It is generally agreed that 100 % testing of large software products is not practicable. Therefore, a variety of quality assurance activities are implemented during software development. The cost of fixing defects varies proportionately with the stage in which the defect is uncovered. A defect uncovered during unit testing costs much less to fix than a defect that is uncovered during system testing stage. The final product is always testable but what is sometimes becomes difficult to test is, the software unit/component. The software product has to be designed and built in such a way that every software unit is independently testable in a stand-alone manner. Testability requirements are normally covered by the standards and guidelines dealing with software architecture, design and construction guidelines.
8. **Interface functionality**—In these days of web based Internet applications, interfacing becomes essential. The internet itself is built with multiple layers. There are many browsers, and different servers, ISPs and networking protocols that an Internet application has to interface with. Additionally, the applications need to be built in such a way that it would be possible to interface with applications that the organization may build later on. This kind of functionality would be covered under standards and guidelines dealing with software design and construction, normally.

2.4 Classification of Requirements Based on Source of Requirements

Yet another way to look at requirements is based on the source from where it is obtained. There are many sources from which we can garner requirements for the proposed software product. Enumerated below are the possible sources for establishing the requirements for a software product.

1. **End users**—These people are those that use the end product to perform their individual business processes. The software product is basically aimed at fulfilling their needs. These people provide the core functionality especially relating to the aspects of inputs, process and outputs at working level. End users may not be able to provide the management requirements expected from the software. End users can be located in the case of project scenario (intended for use within one organization) in the departments funding the software development. In the case of a COTS (Commercial Off The Shelf) product scenario, end users are scattered across the target market for the product. We may perhaps need to conduct market surveys to get their needs or select randomly some end users and interview them to obtain their needs and the core functionality for the product.
2. **Management of customer organization**—These people provide the MIS (Management Information System) portion of the core functionality. They provide what information they need to extract from the software so that they can manage the organization effectively. These may include the special analyses, special reports, audit trails, security concerns, safety concerns and so on, necessary from the software product. In a project scenario, these people can be located in the organization looking at the organization chart. But in the case of COTS product development, we really need to put in efforts to locate such experts. They can be found in the domain experts, academia, and through market surveys selecting the senior management personnel to provide the information.
3. **Domain experts**—These individuals are those that have worked for long years in the business domain in which the proposed software product would be developed. These individuals are especially useful and extensively used in COTS product development. These people may or may not be IT (Information Technology) experts but they would have the knowledge of systems and procedures or the domain. They would know the detailed procedures, formats, templates, guidelines, standards and checklists used by the end users in the domain. Additionally they would be experts in the process that is used to convert the inputs to outputs as well as the legal issues involved with the domain. These people would be occasionally used in project scenarios to obtain information about industry best practices or when the requirements provided by the end users and their management are perceived to be either incomplete or ambiguous. Domain experts can provide end-to-end core functionality or clarify any issues thereof.

4. **Project team**—Project team comprises of the project mangers, project leaders, software designers, business analysts, programmers, testers, User Interface (UI) developers, and Database Administrators (DBAs). These people are also a source of providing requirements albeit the fact that they may not be able to provide core functionality unless the product is proposed to be used in software development activities. They would however be able to provide ancillary functionality such as usability, maintainability, safety, security, and reliability and so on.
5. **Statutes**—Statutes include governmental regulations pertaining not only to usability of the software but also about possible illegal activities. The intended software product shall not either commit, aid or abet any sort of criminal activity. Therefore, the requirements need to include all statutes that need to be implemented as well as ensure that all prohibited functionality is excluded from the proposed software product. The business analysts carrying out requirements analysis ought to be aware of the statutes that mandate inclusion of functionality as well as the functionality that is prohibited. Some of the examples of prohibited activities include stealing of personal data, siphoning away of monies in dormant bank accounts, sending spam emails and so on.
6. **Industry standards**—These include standards of industry (such as ISO, CMMI, NASSCOM (National Software and Services Companies of India) or professional associations (such as IEEE, SPINs, SPMNs) or organizational standards of either the vendor organization or the client organization. These standards address various aspects of software engineering methodologies including processes, guidelines, formats, templates and checklists). A host of information and best practices are available from such standards and ancillary functionality can be derived from those standards.
7. **Software designers**—Software designers can provide ancillary requirements about the efficiency, fault tolerance, operations ease, installation ease, usability, structural stability and so on of the end product. Software designers are also normally part of the project team but are treated separately because software designers play a key role in the final product. Finally it is the software designers that have to shoulder the responsibility for any missing/defective functionality in the end product of software development.
8. **Software programmers**—Software programmers are at the end of the chain for implementing the requirements in the software. However, they are the people who need to implement all the requirements conveyed to them using software design documents. However, coding guidelines, UI guidelines, and any other organizational guidelines would not form part of design documents and programmers are expected to be knowledgeable of such standards as well as implement them effectively in every program they code. Software programmers can specify ancillary functionality aspects pertaining to maintainability, testability, reusability of the code and so on.
9. **Software quality assurance team**—A Software quality assurance team includes reviewers, testers, and process specialists. These people can provide ancillary requirements about testability, and quality perspectives of the proposed software product.

10. **Management of software development team**—These individuals include project manager, project leader and other senior management personnel including program managers. These individuals would be in a position to have a bird's eye view of the overall project and would be able to provide interface requirements to ensure that the software product would be able to interface effectively with other applications in the organization. They would also be able to ensure that all functionality is included for the proposed software product.
11. **Marketing department**—Especially in product development organizations, marketing is a source of product requirements/specifications. A Marketing department can generate product requirements from the field staff or a market survey of the potential users of the proposed product. Market surveys are a very popular vehicle to collect user requirements and freeze product specifications. A Marketing department is the primary source for one-upmanship functionality in a COTS product.

2.5 Levels of Requirements

Institute of Electrical and Electronics Engineers (IEEE) has specified two levels of requirements specifications, namely, the User Requirements Specification and Software Requirements Specification. This gives rise to the question "Do we have two levels of requirements in software development?"

In any product there are two sets of requirements. One is the set of needs that the product fulfills and the second is the set of specifications that the product must adhere to in order for it to fulfill the needs of users. Let us take a bridge over a river (or a body of water) as an example. The need to be fulfilled is a bridge that can carry six lanes of traffic over a river of two furlongs breadth. Now based on this specification, some preliminary work is carried out to determine the depth of the water body, the banking on both the sides, the soil quality at the bottom of the water, the number of cars, trucks and other vehicles that traverse the bridge etc. Based on the study, the specifications for the bridge are finalized. These are the product specifications for the bridge, which include the load it has to support, the type of bridge (suspension, column supported, arched etc.), the width and so on. Based on these product specifications, a bridge is designed. Using the design, the bridge is constructed.

Similarly, in software development, the first expression of the need is the requirement of software to fulfill the requirements of a business process. Taking this first into account, a preliminary study is conducted to ascertain the needs of the business processes. From these results the product specifications would be drawn up.

In the manufacturing industry, the aspect of drawing up product specifications is referred to as Conceptual Design (or the High Level Design) and the working out the details is referred to as the Detail Design (or the Low Level Design).

Thus, there are three layers before the actual fabrication/manufacture/construction begins and these are:

1. The needs
2. The product Specifications
3. The design

In software development too, we have three levels before the coding begins. There are various nomenclatures for these three levels. I am presenting a few here but there could be others.

1. User Requirements Specification (URS), System Requirements Specification (SyRS), Business Function Specification (BFS), Functional Specification (FS), Requirements
2. Low Level Design (LLD), Software Requirements Specification (SRS), Functional Design Specification (FDS), Architecture
3. High Level Design (HLD), Software Design Description (SDD), Software Design Specification (SDS), Detail Design Specification (DDS)

Now the next question that arises is "what requirements do we need to manage—user requirements or software requirements?"

User requirements are original and first in the chain. Software requirements are not original. They are derived from user requirements. When user requirements change, the software requirements also change. Therefore, we need to manage user requirements. If we can minimize changes to user requirements, the changes to software requirements would automatically be minimized. This book focuses on managing user requirements.

2.6 Definition of Requirements in the Context of Software Development

How do you answer the question "what constitutes a software application?"

I am sure there will be numerous or multiple alternative answers for this question. The answer I select is that the application consists of a number of information processing processes. Information processing processes can be further divided into three classes, namely,

1. Input processes
2. Output processes
3. Associative processes

Input processes obtain information from outside the application boundary. The information would be provided by an entity (an individual, a machine or another computer application). The input can be data (facts, and figures about an entity), control data (triggers for events in the application such as start, stop, change, print etc.).

Output processes send information across the application boundary. The recipients of information would be an entity (an individual, a machine or another computer application). The output could be normal data in the form of a report either on paper or computer screen or to another application, or it could be control data in the form of trigger to another application. The giving/receiving application can be on another computer or another machine.

Associative processes are those processes that aid and assist the input and output processes in information processing. Consider a login process; it is neither an input nor an output. Similarly a file upload is a process; and so is an integrity checking process and so is the POD (Power On Diagnostics) process. These are all examples of associative processes.

Requirements elicitation/gathering is mainly enumerating all the processes that form a part of a software application and obtaining details about these processes so that downstream activities can be executed without further reference to the client or with minimal reference to the client.

Now each process has these attributes:

1. Inputs—each process receives certain inputs. While input processes receive inputs from external sources, output processes and associative processes receive inputs from the internal sources.
2. Outputs—Each process delivers some outputs. While output processes deliver to external recipients, input processes and associative processes deliver to internal recipients.
3. Process—each process carries out some transformation of inputs and converts them to outputs. Each process consists of some related steps with a start and an end event. The process includes verification of the inputs for their completeness, appropriateness and freedom from errors.
4. Triggers—each process needs a trigger that initiates the process into execution. The trigger could be an event initiated by an individual or another application or could even be by the application itself.

When we enumerate all the processes and define all the above four aspects for each of the processes comprising the software application, we have a complete requirements specification document.

2.7 Evolution of Requirements

Requirements start out as a single idea and over a period of time, evolve into a full set which can then be further processed into a software product. The phases in the evolution of requirements are discussed in this section. Please note that it is not mandatory that every organization uses these phases; some of the phases may be dropped or some other phases may be used; or they may use different set of phases altogether.

In a new product—non-existent in the market—This is a scenario in which a completely new product, the type which is not existing in the market is contemplated. In this scenario, the requirements are evolved as follows:

1. **Idea germination**—Here, the entrepreneur or the product manager germinates an idea based on his/her observations of the needs of the target market and perceives a need for a product that can fulfill the unfulfilled needs of target customers. In a large organization, there could be a few product managers and all of them could come up with new product ideas. It is stated that it takes about eight serious ideas to get one product idea to be approved and built. Not all approved ideas are successful and not all dropped ideas are bad ideas. Remember Chester Carlson was turned down for 5 years by organizations such as IBM and GE when he approached them for his idea on Xerox machines? This approved idea is the first phase in the evolution of requirements for developing new product the kind of which is non-existent in the market.
2. **Brainstorming**—Brainstorming is a technique in which experts in the subject at hand gather in an informal meeting and give a free rein to their imagination. All ideas expressed are recorded for later analysis which would shortlist ideas that are worthy of pursuing. In requirements management, all desirable product features are enumerated by the brainstorming, which are then sifted and feasible ones are culled. These form the initial requirements for the product.
3. **Market/customer/consultant surveys**—Now the initial requirements are tested using a market survey. Various methods of market survey are available and an appropriate one would be selected and used. The market survey would validate the initial requirements and usually adds a few more requirements. These requirements would be further validated by experts drawn from the market, consultants or academia using personal interviews.
4. **Personal interviews**—Personal interviews are conducted with selected experts who may be marketers, product designers, support staff, consumers, consultants or academics. The requirements finalized by the market surveys would be discussed with them for two purposes. One—to validate the requirements; two—to add to the requirements. This is the final step before attempting to design a prototype and go to market again to validate the product.
5. **Prototype and demos**—After the requirements are validated through personal interviews, normally a prototype of the product would be built. Prototypes are discussed in greater detail in Chap. 3. Now these prototypes are shown to prospective customers, and experts in the field. Their feedback is taken, evaluated and requirements are updated. This is the final step in the evolution of requirements.
6. **Freeze requirements**—Freezing the requirements involves documenting the requirements conforming to organizations standards and subjecting further changes to the rigor of configuration control and change management. The frozen requirements are then used to carry out full scale product design and development of the product and introducing it into the market. Normally, the

changes proposed on frozen requirements are considered for the next upgrade/release of the product.

New product—existing in the market—This is a scenario in which the product is new to the organization proposing to develop it but something similar is available in the market. When somebody or an organization wishes to develop a competitor to a product that is already existing in the market, the requirements evolve through the following phases:

1. **Idea germination**—The entrepreneur or the product manager or someone with a say gets an idea to develop a product as a competitor for an existing product. The existing product may not be fulfilling the market expectations or the market is large enough to accommodate a new and innovative product or some such motive could be behind the idea. This is the preliminary requirement.
2. **Market/customer/consultant surveys**—Market surveys are conducted to confirm the need for an additional product and to unearth the needs unfulfilled by products existing in the market.
3. **Personal interviews**—Personal interviews with experts in the field are conducted to validate the data from market surveys and to add any more requirements to the list.
4. **Brainstorming**—In the brainstorming, one-upmanship ideas over the existing product are generated. Ideas about more functionality, better presentation, better workflow, improved ease-of-use, more user options, flexibility etc. are generated during brainstorming. These would be analyzed and requirements are finalized.
5. **Prototypes**—Normally prototypes are not constructed in this scenario as a working product is available in the market. But, it may be used sometimes to prove a concept or a feature and get feedback from the market.
6. **Freeze requirements**—Freezing requirements as noted earlier involves approving the requirements document and subjecting it to the rigor of configuration control and change management.

Product upgrade—We have a product that has been in the market for some time and we have been receiving feedback about the desired additional features or improvements in the existing features from our customers, field support staff, marketers and Value Added Resellers (VARs). We also find that competitors have brought their products into the market which are cutting into our market share. Therefore, we wish to upgrade our product to keep it competitive and attractive to the market. Here is how the requirements evolve:

1. **Feedback/Surveys from VARS**—VARS are one valuable source of feedback about the existing product owing to their proximity to the competitors in the market and end users. Whenever they provide feedback, we need to analyze and resolve it. When we contemplate upgrading the product, we need to conduct a survey to elicit their views on the improvements desirable to our product over and above what they already communicated. The information obtained from the

feedback and survey of VARS needs to be analyzed to remove duplication and the feasibility of implementation.

2. **Feedback/Surveys from tech support staff**—Support staff is in the field and are the first contact with the customers/end users. They would receive issues, concerns and suggestions for improvement from the customers. Most product organizations would have a formal mechanism to capture this kind of information during their interaction with customers. When we contemplate a product upgrade, we need to collate all such feedback and conduct a formal survey to elicit any further suggestions from the field staff to comprehensively capture all the expertise gained by the field support executives. The feedback and survey results can be analyzed to finalize upgrade requirements.
3. **Customer/market surveys**—Customers are the only people in the supply chain that can provide first-hand feedback. All others can only provide second-hand feedback. Therefore, we need to conduct customer surveys. These would validate the feedback obtained from VARs and field support staff.
4. **Personal interviews**—We need to conduct personal interviews to validate the findings of various surveys conducted as well as to uncover any biases or prejudices that have crept into our survey results. We conduct personal interviews on a sampling basis using stratified sampling technique.
5. **Freezing of requirements**—Once we have collected feedback/survey results from VARS, field support staff and customers, we analyze the results to consolidate the requirements and eliminate duplicates. We select the requirements for the product upgrade based on their feasibility. Then we document the requirements conforming to organizational standards and subjecting the document to the rigor of configuration and change management.

Project development scenario—The end result of software development either in the product development or project development is a software product. Then why should we distinguish product development and project development? The way I distinguish project development from product development is based on the use of the end product. In the *project development* scenario, the end product is proposed to be used by one customer or one set of end users within one organization. The end product of *product development* on the other hand, is proposed to be used in multiple locations, in multiple organizations, and by different sets of end users. Because of this distinction, the process of obtaining and finalizing requirements assumes different levels of importance. In a product development scenario, we need to spend much more time on finalizing requirements because any mistake committed during this phase would have a strategic impact on the final success of the product and the survival of the organization itself. Since the project is for one customer, the preferences of that customer assume paramount importance. In the project development scenario, we do not have VARS or field support staff or multiple customers to cater to. We have only one customer and one set of end users to whom we should listen to and satisfy them through our software product. In this scenario, we have two classes, namely in-house project and outsourced project. In the in-house project, the end product would be used within the

same organization, even if it is in another department. An outsourced project is one in which the end product would be used in an organization other than the one in which it was developed. In outsourced projects, the organization that develops the product would normally be specializing in software development for other organizations. In both scenarios, the business analysts would interact with the end users performing business processes, to elicit, gather and finalize requirements. The evolution of requirements in these two scenarios is discussed below.

In-house project—new project—The evolution of requirements in in-house new project would take this course:

1. **Proposal by a functional department**—A department responsible for a set of business processes proposes computerization of the processes with a view, perhaps to increase efficiency, reduce turnaround time or improve quality and initiates the process. The first step in the process is the definition of the scope of work and defining the boundaries of the application that is proposed for computerization. This is the initial requirement. This scope would then be forwarded to the IS (Information Systems) department to come up with a budget. The department obtains financial approval for the budget requested by the IS department and requests the IS department to carry out all necessary activities to computerize the selected set of business processes. The IS department may be called by different names in different organizations.
2. **User Requirements definition**—Business Analysts from the IS department would elicit/gather requirements from the end users. Elicitation and gathering of requirements is described in Chap. 3 of this book. The principal methods used in this scenario are personal interviews and study of procedure manuals, forms and templates used in the performance of the processes as well as personal observation to collate the user requirements. All the requirements collated are analyzed to finalize requirements for the proposed system. Requirements Analysis is described in Chap. 4 of this book.
3. **Finalization of the requirements**—The requirements allocated to the proposed system are then documented conforming to the organizational standards. This document is internally reviewed and approved within the IS department and is then forwarded to the functional department for review and approval. The functional department reviews the document and requests clarifications, if necessary. Normally a meeting takes place between the executives of the functional department and the IS department, in which the document contents are discussed and clarifications from both sides are provided. The IS department implements any feedback received from the functional department and obtains the approval from the functional department. This document acts as the reference between the IS department and the functional department during the course of the software development. This document would be subjected to the rigor of the configuration and change management.

In-house project—upgrade—Usually, the software products being developed and used inside an organization are maintained regularly as needed. The strict meaning of "maintenance" is to restore/keep an artifact/product to its working

state. However, in the software industry, product expansion is also carried out under the caption of "maintenance". But such product expansions, carried out as part of software maintenance, are usually minor in nature. When the product is scaled up significantly, an upgrade project is undertaken. Such major upgrades are necessitated from time to time and could be to use the newer technologies (such as web enabling the applications) or newer business scenarios (such as mergers or acquisitions or a major re-organization, etc.) For an upgrade project undertaken in the organization, in-house, the requirements would evolve in the following manner.

1. **Proposal by the functional department**—The trigger for the functional department to propose a major upgrade to the software product would be external usually. The trigger may be availability of new technology, availability of sparable funds, a change in the business scenario and so on. In such cases, the functional department would define the scope of work for the product upgrade and invite the IS department to propose a budget and obtains the financial approval for the budget. The definition of the scope of work for the expansion would be the initial set of requirements.
2. **Use Requirements definition**—Business Analysts would approach the executives of the functional department and elicit/gather the requirements. In this case, the elicitation/gathering is limited the expanded functionality only. If the upgrade is to convert the existing application from an existing platform to a newer platform, this phase may be avoided altogether except to see if any additional requirements are necessitated due to the change in the technical platform. These requirements are analyzed and requirements are finalized for the project.
3. **Finalization of the requirements**—These requirements are documented and are forwarded to the functional department for approval. The functional department accords approval after getting its feedback implemented in the document by the IS department. This document is then subjected to the rigor of the organizational configuration and change management.

Outsourced project—new project—The outsourced project in this context is the project executed at the vendor's place for a client. An organization realized the need for computerization of a set of its business processes, but decided to outsource the software development portion of the project to a specialist software development organization. Now, this project is proposed to be executed at the organization specializing in software development. The requirements evolve in the following manner in this scenario.

1. **Project acquisition**—The first step in the project execution is to acquire the project from an outsourcer organization. The projects are outsourced in different combination of software development phases, such as (a) Requirements, design, construction, testing and delivery; (b) Design, construction, testing and delivery; (c) Construction, testing and delivery; (d) Testing and delivery and any other combination. The projects may be on fixed-price contracts or time-

and-material priced contracts. Whatever the case may be, the project acquisition phase would certainly include the definition of the scope of the project, which is the initial requirement for the project.

2. **Requirements elicitation/gathering**—During this phase, Business Analysts from the vendor organization (or internal analysts if requirements definition is not outsourced) would elicit/gather the requirements from the client executives. They would be using personal interviews and surveys to elicit the requirements and study the existing process manuals, formats and templates to gather the requirements. They would then be analyzed to form the project requirements. Sometimes the requirements may be provided by the client along with the purchase order. In such cases, the organization just needs to assess the received requirements for adequacy and completeness.
3. **Analyze the requirements**—Subject the collated requirements to analysis to determine their technical feasibility, eliminate duplicates, group them into logical groups and prioritize them, Analysis is described in Chap. 4 of this book.
4. **Finalize the requirements**—This involves documenting the analyzed requirements to ensure that they are conforming to the agreed standards and obtaining internal approvals after due quality assurance process of the organization.
5. **Customer approvals**—The finalized requirements would then be submitted for customer review and approval. The customer would review the requirements document and request improvement, if necessary, by providing the feedback. When the customer is satisfied, approval would be accorded to the requirements document. Now, this approved requirements document would be brought under the vendor's configuration and change management process. This document would be the reference point for all interaction between the customer and the vendor in matters relating to project requirements.

Outsourced project—upgrade—Outsourcing a major product upgrade to a vendor is risky as the source code needs to be provided to the vendor to upgrade the product. It had to be done in the case of Y2 K conversion. Millions of LOC (Lines of Code) had to be uploaded to vendor's computers for them to convert it to be compliant with Y2 K requirements. No major breach-of-trust case was reported arising out of stealing intellectual property contained in the source code even though most of the upgrade is carried out overseas. In cases where the upgrade is to re-develop the code on a new platform, of course, the original source code need not be given to the vendor. Requirements in this scenario would evolve in this manner:

1. **Project acquisition**—In this scenario, the definition of scope would be much more lucid than in the scope definition of a new project. This is due to the fact that the product is working and the upgrade requirements are better defined. It is usual to get a list of detailed requirements in the RFP (Request for Proposal) itself. However, in platform upgrade, requirements may have to be elicited/gathered from the customer executives. The RFP provides the initial requirements.

2. **Requirements elicitation/gathering**—When required, the requirements need to be gathered in the same way as explained for the new project requirements evolution in the above section. When requirements need to be elicited/gathered, they will be collated using personal interviews and surveys as the tools.
3. **Requirements analysis**—When requirements are provided, they will be analyzed for their adequacy and completeness and clarifications are obtained. If there are requirements which are elicited/gathered, they will be analyzed for their feasibility for implementation.
4. **Finalize the requirements**—The finalized requirements would then be documented conforming to the agreed standards. These will be subjected to an internal quality assurance process and internal approvals need to be obtained. Then the approved document would be forwarded to the customer for their feedback and approval.
5. **Customer approvals**—A Customer would review the requirements document for the completeness of information and to ensure that the requirements are properly understood by the vendor. The customer would communicate feedback, if any, for implementation and after the document is to their satisfaction, the customer would accord the approval to the requirements document. This approved document would be subjected to the rigor of the vendor's configuration and change management process. This document would be the reference point for all matters relating to project requirements between the outsourcer and the vendor.

Of course, there would be variations to the evolution of requirements from one organization to another. What is presented here are the typical scenarios. Another aspect to be noted is that the projects are of various types. There are full life cycle projects, part life cycle projects, testing projects, conversion projects, porting projects, migration projects and so on.[1] Whatever the project type, requirements are the first step in the project execution and any error committed in this phase would have a recurring impact on all the subsequent phases of the project.

[1] Interested readers are suggested to read the book "Mastering Software Project Management: Best Practices, Tools and Techniques" by Murali Chemuturi and Thomas M. Cagley, Jr., published by J.Ross Publishing, Inc, USA, 2010.

http://www.springer.com/978-1-4614-5376-5

Requirements Engineering and Management for Software Development Projects

Chemuturi, M.

2013, XXII, 266 p., Hardcover
ISBN: 978-1-4614-5376-5