DOWNLOAD PDF SOFTWARE SPECIFICATION AND DESIGN

Chapter 1 : Software Specification and Design: An Engineering Approach - CRC Press Book

Software Specification and Design: An Engineering Approach offers a foundation for rigorously engineered software. It provides a clear vision of what occurs at each stage of development, parsing the stages of specification, design, and coding into compartments that can be more easily analyzed.

A design document outlines an implementation that will satisfy the requirements of the SRS. There are many aspects of the system that need to be designed: Division of the system into components Communications, APIs and dependencies between components Classes, attributes. The SRS describes precisely what the system will do. It is based on user needs It sets the stage for design and system testing Customers should be able to review the SRS and help validate it. Validation of requirements is checking the written requirements against unwritten reality Design outlines how the system will work. It is based on the SRS It sets the stage for implementation and integration testing Usually only members of the development team need to review the design. What Do They Have in Common? Both SRS and Design documents are technical documents that need to be precise. In fact, sometimes the same notation is used for multiple purposes Both need to be reviewed by several stakeholders and defects must be corrected. Where Do They Fit in the Process? V-shaped waterfall User needs gathers information for the SRS The SRS describes the system that will satisfy some user needs, sets the stage for design, and will be tested in system test. The design documents outline how the system will implement the features specified in the SRS and sets the stage for integration test. What Makes a Good Specification? Short requirements are more understandable and maintainable. And, they can often be more precise when you use formalisms. Incorrect requirements will lead to reduced revenue, or major rework. Incomplete requirements also leads to major rework and last-minute schedule changes. The specification should not bias the design or implementation. You may come up with better ideas for the design later, you do should not have to change the SRS unless your goals for the product change.

Chapter 2 : Software design - Wikipedia

Document Outline. Here is the outline of the proposed template for software design specifications. Please note that many parts of the document may be extracted automatically from other sources and/or may be contained in other, smaller documents.

Interacts exclusively with the user. Uses the class library package defined above. Communicates with the server application through the server procedure proxy class contained in the class library. Reporting Application Uses Jasper Reports to present html-based, web-accessible reports based on the data found in the database. These tools will allow the various users to accomplish the management of the software, and the management of the rest of their projects. Login Screen - This first screen will allow Steve Techie to protect access to the rest of the application. Employee Management Screen - This screen allows Steve Techie to enter new employees, who can then be assigned to projects. Reports Screen - This screen allows Steve Techie to generate reports on project and employee activity, which will be of great interest to the owner "Mr. Manager" , and perhaps the customer "Bob PaysUsWell". The reports types are: It will have the following layout: Login Screen - This lets any user log in to the Project Server. Projects Overview Screen - After login, the application will present this screen and will list all of the projects that the client program knows are assigned to the current user. This may be incomplete or out of date if the user has not synchronized with the Project Server recently. The client will automatically attempt to synchronize after login. During this synchronization a potential conflict can occur if a project has been altered both server side and client side during the time the client was offline. The user will be asked to resolve such conflicts before being allowed to go onto other tasks. A status display showing whether the application is online with the Project Server or not, and the last date and time when a successful synchronization occurred. A tabbed list of the projects: Any of the lists may be sorted by priority. A user may, by double clicking on a project in the list, go to the Project Maintenance Screen for that project. This will go to a blank Project Maintenance Screen for that project. Project Maintenance Screen - This screen contains information about a particular project, and buttons to investigate further details. It has three main tabs. The first tab, "Summary" contains general data in the top two thirds of the screen--for a full list of those fields see the mock up of the screen in section 7. Action Items are steps to be taken in the completion of the project. These can be marked as finished when the user completes them. When viewing the action items lists, the user may click on an action item to view further details, or to convert the action item to a project. In this latter case, the newly created project will remain in the action items list but will have a modified look, and the details screen for that action item will be another Project Maintenance Screen instead of a simpler action item detail. This feature may be used by the SuperOwner "Veep McBleep" , in order to delegate further tasks to his subordinates. The second tab of the Project Maintenance Screen, "Contact Information", will show contact information for key people related to the project as further proof against communication breakdown. The third tab, "Extended Documentation" will contain a detailed description of the project, and links to related resources. Time Entry Screen - This will be a very simple screen which allows the user to specify a project, a start time, and an end time. When the values are entered, the user clicks the "Check in" button, and the time report is added to the project. This will allow the average user "Joe Schmoe" , to account for his time. These time reports will be compiled into reports for Mr. Manager using the Employees and Reports Application. Also, a tab will allow a user to account for costs accrued during the course of their duties. It will contain most of the basic functionality of the PC Client. The issues related to the PDA client are ones of scale. Some of the screens will need to be modified for easy viewing on the PDA screen. Here are the screens and how they might be affected: Data groups will be replaced by buttons which will bring up more detailed screens for those data groups.

Chapter 3 : Embedded System Experts, Software-Hardware Development | MicroTools Inc, CT, Connecticu

Introduction Purpose of this document. The Software Design Specification (SDS) document will have two major releases: Version 1 focuses on specifying a high-level view of the architecture of our system, and on the interaction between the user and the system.

Difference between a System Specification and a Software Specification? The key to this blog posting is to get a complete understanding of Software requirements specifications, not technical specifications which are easy to confuse. Here is the difference. It should care less about implementation. A technical specification describes the internal implementation of the software. It talks about data structures, relational database models, choice of programming protocols, and environmental properties. So before we continue, please remember that this blog post is about Software requirements specifications as opposed to technical specifications. While they both define behavior, the use case tells the story showing the end-to-end scenario. To further define, a use case defines a goal-oriented set of interactions between external actors and the system under consideration. Actors are parties outside the system that interact with the system. A primary actor has the goal requiring the assistance of the system. A secondary actor is one from which the system needs assistance. Generally, use case steps are written in an easy-to-understand structured story using the verbage of the domain. This is engaging for users who can easily follow and validate the use cases, and the accessibility encourages users to be actively involved in defining the Software requirements specifications. Software Requirements and Specifications 2. A key benefit of developing a software requirement specification is in streamlining the development process. The developer working from the software requirement specification has, ideally, all of their questions answered about the application and can start to develop. Since this is a functional specfication that was client approved, they are building nothing less than what the customer wants. There should be nothing left to guess or interpret when the functional spec is completed. This is the brilliance of the software requirement specification. The customer or product team define most software requirements in functional terms, leaving design and implementation details to the developers. They may specify price, performance, and reliability objectives in fine detail, along with some aspects of the user interface. Because there are so many types of requirements, the term requirement is odd because it describes the concept of an objective or goal or necessary characteristic, but at the same time the term also describes a kind of formal documentation, namely the requirements document. Putting aside the particular document for now, Software requirements specifications are instructions describing what functions the software is supposed to provide, what characteristics the software is supposed to have, and what goals the software is supposed to meet or to enable users to meet. A set of Software requirements specifications would include documents spelling out the various requirements for the project -- the "what" -- as well as specifications documents spelling out the rules for creating and developing the project -- the "how" or implementation specifications. Project requirements provide an obvious tool for evaluating the quality of a project, because a final review should examine whether each requirement has been met. Requirements tend to change through the course of a project, with the result that the product as delivered may not stick to the available Software requirements specifications -- this is a disturbing part of the quality assurance process. A software requirement specification document describes how something is supposed to be done. A specifications document may list out all of the possible error states for a certain form, along with all of the error messages that should be displayed. The specifications may also describe the steps of any functional interaction, and the order in which they should be followed by the user. A requirements document, on the other hand, would state that the software must handle error states reasonably and effectively, and provide explicit feedback to the users. The specifications show how to meet this requirement. Specifications may take several forms. They can be a straightforward listing of functional attributes, they can be diagrams or schematics of functional relationships or flow logic, or they can occupy some middle ground. Software requirements specifications can also be in the form of prototypes, mockups, and models. Project specifications are much more important for determining the quality of the product. Every rule and functional relationship provides a test point. A disparity between the program and its specification is an error in the

program if and only if the software requirement specification exists and is correct. A program that follows a terrible specification perfectly is terrible, not perfect. You will want someone who is very familiar with GUI issues and web design, familiar enough with technology to know its limitations and capabilities, and someone who is a very skilled and a good writer. While writing a spec, you will spend a great deal of time imagining how a user might use a certain feature and how they may navigate their way through the software. Examples of Software Requirements and Specifications The following list describes the kinds of documents that belong to the body of requirements and specifications document. These are not all required for every software project, but they do all provide important information to the developers, designers and project managers tasked with implementing a project and to the quality assurance people and testers responsible for evaluating the implementation of the project. It is encouraged that any user requirements document define and describe the end-user, and that any measurements of quality or success be taken with respect to that end-user. User requirements are usually defined after the completion of task analysis, the examination of the tasks and goals of the end-user. First, it can refer to the software requirements that describe the capabilities of the system with which the product will function. For example, the web site may need to run on an application server and be clustered. Second, it can refer to the requirements that describe the product itself, with the meaning that the product is a system. There are two categories of system requirements. Software requirements specifications specify what the system must do. User requirements specify the acceptable level of user performance and satisfaction with the system. For this later definition, it is preferred to use the more general term "requirements and specifications" over the more boring "system requirements". If the user requirements are written for the requestor and not the end-user, the user requirements are combined with the Software requirements specifications. These specifications are typically used to build the system exclusive of the GUI. With respect to a web site, a unit is the design for a specific page or category of page, and the software requirement specification would detail the functional elements of that page or page type. For example, the design for the page may require the following functions: A component is a set of page states or closely related forms of a page. For example, a component might include a submission page, and the acknowledgement page. Designing flowcharts can be very handy when trying to work through a lot of information. Also include the navigational path through the information on this diagram. Visio is an awesome tool for developing technical diagrams and flow charts such as these. Create a prototype A prototype of a web application is a set of static html pages put together to show the key pages of the application and the GUI. It allows the software requirement specification writer to have something close to a working model to test their ideas out on, start focusing on the layout, and begin gathering input about the look and feel. In the iterative process, it helps to have this grand view look at a potential finished product while also seeing how the components of the system affect the overall product. Instead of boxes and arrows, bulleted comments, and a list of functions, you have an actual web page to look at and experience. Whether you create a prototype now or later, you should definitely have one before you start writing the actual software requirement specification. Aside from being a convenience for you and allowing you to possibly iterate your design much faster, it allows your team members and stakeholders the opportunity to look at the application and deliver much more specific and useful input about what you are doing. Not everyone can look at an informational flowchart and realize something is way they want it. Most people can when looking at a prototype. You are laying the groundwork for what will become the structure of your application. Create A Design Document As stated in the previous section, a design document serves to collect everything we know about the application at this point into a kind of pre-software requirement specification document and allow people the opportunity to review it and provide feedback. You will want to use this document as a tool for building consensus and, simultaneously, managing the expectations of people about what is on the way. The key thing about design documents that you want to keep in mind is that they say little about the visual appearance of the application aside from layout , and they say very little about the specifics of the user experience. If the software requirement specification is at the ground level, then this design is somewhere in the neighborhood of a higher level. The developers work from the Software requirements specifications, and translate the functionality into their actual coding implementation and methodologies. By the time the programmers start the dev process, all issues need to be resolved. As you write

the specification you may think of useful factoids that will be helpful to just one of those groups. For example, I flag messages to the engineer, which usually describe some technical implementation detail, as "Tech Notes". Marketing people ignore those. Some programming teams adopt a "waterfall" mentality: This approach is why most Software requirements specifications have such a bad reputation. Specifications should be updated frequently. The updating continues as the product is developed and new decisions are made. The software requirement specification always reflects our best collective understanding of how the product is going to work. The software requirement specification is only frozen when the product is code complete The software requirement specification should not be released daily. Congruency between the customers stakeholders and the suppliers on what the software product is to do. The complete description of the functions to be performed by the software specified in the Software Requirement Specification will help the potential users to determine if the software in question adheres to their needs. Reduce the development effort. Careful review of the requirements in the Software Requirement Specification can reveal omissions, misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct. Provide a basis for estimating costs and schedules. The description of the product to be developed as given in the Software Requirement Specification is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates. Provide a baseline for validation and verification. Organizations can develop their validation and Verification plans much more productively from a good Software Requirement Specification. As a part of the development contract, the SRS provides a baseline against which compliance can be measured. The Software Requirement Specification makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organization, and suppliers find it easier to transfer it to new customers. Serve as a basis for enhancement. Because the Software Requirement Specification discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation.

Chapter 4 : Software Requirements Specification, what you NEED to know

Software Specification and Design: An Engineering Approach presents a basis for rigorously engineered software. It supplies a transparent imaginative and prescient of what happens at every stage of improvement, parsing the levels of specification, design, and coding into compartments that may be extra simply analyzed.

Overview[edit] Software design is the process of envisioning and defining software solutions to one or more sets of problems. One of the main components of software design is the software requirements analysis SRA. SRA is a part of the software development process that lists specifications used in software engineering. If the software is "semi-automated" or user centered , software design may involve user experience design yielding a storyboard to help determine those specifications. If the software is completely automated meaning no user or user interface , a software design may be as simple as a flow chart or text describing a planned sequence of events. There are also semi-standard methods like Unified Modeling Language and Fundamental modeling concepts. In either case, some documentation of the plan is usually the product of the design. Furthermore, a software design may be platform-independent or platform-specific , depending upon the availability of the technology used for the design. The main difference between software analysis and design is that the output of a software analysis consists of smaller problems to solve. Additionally, the analysis should not be designed very differently across different team members or groups. In contrast, the design focuses on capabilities, and thus multiple designs for the same problem can and will exist. Depending on the environment, the design often varies, whether it is created from reliable frameworks or implemented with suitable design patterns. Design examples include operation systems, webpages, mobile devices or even the new cloud computing paradigm. Software design is both a process and a model. The design process is a sequence of steps that enables the designer to describe all aspects of the software for building. Creative skill, past experience, a sense of what makes "good" software, and an overall commitment to quality are examples of critical success factors for a competent design. It begins by representing the totality of the thing that is to be built e. Similarly, the design model that is created for software provides a variety of different views of the computer software. Basic design principles enable the software engineer to navigate the design process. Davis [3] suggests a set of principles for software design, which have been adapted and extended in the following list: The design process should not suffer from "tunnel vision. The design should be traceable to the analysis model. Because a single element of the design model can often be traced back to multiple requirements, it is necessary to have a means for tracking how requirements have been satisfied by the design model. The design should not reinvent the wheel. Systems are constructed using a set of design patterns, many of which have likely been encountered before. These patterns should always be chosen as an alternative to reinvention. Time is short and resources are limited; design time should be invested in representing truly new ideas by integrating patterns that already exist when applicable. The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world. That is, the structure of the software design should, whenever possible, mimic the structure of the problem domain. The design should exhibit uniformity and integration. A design is uniform if it appears fully coherent. In order to achieve this outcome, rules of style and format should be defined for a design team before design work begins. A design is integrated if care is taken in defining interfaces between design components. The design should be structured to accommodate change. The design concepts discussed in the next section enable a design to achieve this principle. The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered. Well-designed software should never "bomb"; it should be designed to accommodate unusual circumstances, and if it must terminate processing, it should do so in a graceful manner. Design is not coding, coding is not design. Even when detailed procedural designs are created for program components, the level of abstraction of the design model is higher than the source code. The only design decisions made at the coding level should address the small implementation details that enable the procedural design to be coded. The design should be assessed for quality as it is being created, not after the fact. A variety of design concepts and design measures are available to assist the designer in assessing quality throughout the development process. The design should

be reviewed to minimize conceptual semantic errors. There is sometimes a tendency to focus on minutiae when the design is reviewed, missing the forest for the trees. A design team should ensure that major conceptual elements of the design omissions, ambiguity, inconsistency have been addressed before worrying about the syntax of the design model. Design Concepts[edit] The design concepts provide the software designer with a foundation from which more sophisticated methods can be applied. A set of fundamental design concepts has evolved. They are as follows: Abstraction - Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose. It is an act of Representing essential features without including the background details or explanations. Refinement - It is the process of elaboration. A hierarchy is developed by decomposing a macroscopic statement of function in a step-wise fashion until programming language statements are reached. In each step, one or several instructions of a given program are decomposed into more detailed instructions. Abstraction and Refinement are complementary concepts. Modularity - Software architecture is divided into components called modules. Software Architecture - It refers to the overall structure of the software and the ways in which that structure provides conceptual integrity for a system. Good software architecture will yield a good return on investment with respect to the desired outcome of the project, e. Control Hierarchy - A program structure that represents the organization of a program component and implies a hierarchy of control. Structural Partitioning - The program structure can be divided into both horizontally and vertically. Horizontal partitions define separate branches of modular hierarchy for each major program function. Vertical partitioning suggests that control and work should be distributed top down in the program structure. Data Structure - It is a representation of the logical relationship among individual elements of data. Software Procedure - It focuses on the processing of each module individually. Information Hiding - Modules should be specified and designed so that information contained within a module is inaccessible to other modules that have no need for such information. In his object model, Grady Booch mentions Abstraction, Encapsulation, Modularisation, and Hierarchy as fundamental software design principles. The importance of each consideration should reflect the goals and expectations that the software is being created to meet. Some of these aspects are: Compatibility - The software is able to operate with other products that are designed for interoperability with another product. For example, a piece of software may be backward-compatible with an older version of itself. Extensibility - New capabilities can be added to the software without major changes to the underlying architecture. Modularity - the resulting software comprises well defined, independent components which leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a desired software system. This allows division of work in a software development project. Fault-tolerance - The software is resistant to and able to recover from component failure. Maintainability - A measure of how easily bug fixes or functional modifications can be accomplished. High maintainability can be the product of modularity and extensibility. Reliability Software durability - The software is able to perform a required function under stated conditions for a specified period of time. Reusability - The ability to use some or all of the aspects of the preexisting software in other projects with little to no modification. Robustness - The software is able to operate under stress or tolerate unpredictable or invalid input. For example, it can be designed with resilience to low memory conditions. Security - The software is able to withstand and resist hostile acts and influences. Default values for the parameters must be chosen so that they are a good choice for the majority of the users. Portability - The software should be usable across a number of different conditions and environments. Scalability - The software adapts well to increasing data or number of users. Modeling language[edit] A modeling language is any artificial language that can be used to express information, knowledge or systems in a structure that is defined by a consistent set of rules. These rules are used for interpretation of the components within the structure. A modeling language can be graphical or textual. Examples of graphical modeling languages for software design are: Flowchart is a schematic representation of an algorithm or step-wise process. Jackson Structured Programming JSP is a method for structured programming based on correspondences between data stream structure and program structure. Unified Modeling Language UML is a general modeling language to describe software both structurally and

behaviorally. It has a graphical notation and allows for extension with a Profile UML. Alloy specification language is a general purpose specification language for expressing complex structural constraints and behavior in a software system. It provides a concise language base on first-order relational logic.

DOWNLOAD PDF SOFTWARE SPECIFICATION AND DESIGN

Chapter 5 : Software requirements specification - Wikipedia

A software requirements specification (SRS) is a technical document that describes in detail the externally visible characteristics of a software product.

Get Your Copy Here Practical Tips For A Improve Ebook Reading Most of the times, it has been believed that the readers, who are using the eBooks for first time, happen to really have a rough time before getting used to them. Most often, it happens when the brand new readers stop using the eBooks as they are unable to utilize all of them with the proper and effective fashion of reading these books. There present number of motives behind it due to which the readers stop reading the eBooks at their first most effort to make use of them. Nonetheless, there exist some techniques that could help the readers to have a good and powerful reading experience. Someone should adjust the proper brightness of screen before reading the eBook. It is a most common problem that almost all of the folks generally tolerate while using an eBook. Due to this they suffer with eye sores and headaches. The best alternative to overcome this acute difficulty is to decrease the brightness of the screens of eBook by making specific changes in the settings. A great eBook reader should be installed. It will be helpful to really have a great eBook reader in order to truly have a good reading experience and high quality eBook display. You can also make use of complimentary software that may provide the readers that have many functions to the reader than only an easy platform to read the wanted eBooks. You can also save all your eBooks in the library that is additionally supplied to the user by the software program and have a good display of all your eBooks as well as get them by identifying them from their unique cover. Aside from offering a place to save all your precious eBooks, the eBook reader software even give you a great number of features to be able to improve your eBook reading experience than the standard paper books. You can even enhance your eBook reading experience with help of choices furnished by the software program like the font size, full display mode, the particular variety of pages that need to be shown at once and also alter the color of the background. You must not make use of the eBook continuously for several hours without rests. You must take appropriate rests after specific intervals while reading. Nonetheless, this will not mean that you need to step away from the computer screen every now and then. Continuous reading your eBook on the computer screen for a long time without taking any break can cause you headache, cause your neck pain and suffer with eye sores and in addition cause night blindness. So, it is necessary to give your eyes rest for some time by taking breaks after particular time intervals. This can help you to prevent the troubles that otherwise you may face while reading an eBook constantly. While reading the eBooks, you should prefer to read enormous text. So, raise the size of the text of the eBook while reading it on the display. Despite the fact that this can mean that you will have less text on every page and greater number of page turning, you will have the ability to read your wanted eBook with great convenience and have a good reading experience with better eBook display. It is suggested that never use eBook reader in full screen mode. It is suggested not to go for reading the eBook in fullscreen mode. Though it may seem simple to read with full-screen without turning the page of the eBook fairly frequently, it place ton of stress on your eyes while reading in this mode. Constantly prefer to read the eBook in the exact same span that would be similar to the printed book. This really is so, because your eyes are used to the span of the printed book and it would be comfy for you to read in exactly the same manner. By using different techniques of page turn you can additionally boost your eBook experience. Check out whether you can turn the page with some arrow keys or click a certain portion of the screen, apart from using the mouse to handle everything. Attempt to use the mouse if you are comfortable sitting back. Lesser the movement you have to make while reading the eBook better is going to be your reading experience. This will help to make reading easier. By using each one of these powerful techniques, you can surely boost your eBook reading experience to a terrific extent. These tips will help you not only to prevent certain risks which you may face while reading eBook regularly but also ease you to take pleasure in the reading experience with great comfort. An Engineering Approach mediafire. An Engineering Approach pdf, epub, docx and torrent then this site is not for you. The download link provided above is randomly linked to our ebook promotions or third-party advertisements and not to download the ebook that we reviewed. We

recommend to buy the ebook to support the author. Thank you for reading.

Chapter 6 : ICS Specification vs. Design

Clearly demonstrates how to tackle the difficult task of software specification and design. Focusing on the specification to design transition, it provides step-by-step rules, guidelines, heuristics, hints and tips.

Of course you want the specification to be correct. No one writes a specification that they know is incorrect. Unambiguous â€" An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. Again, easier said than done. Spending time on this area prior to releasing the SRS can be a waste of time. But as you find ambiguities â€" fix them. Complete â€" A simple judge of this is that is should be all that is needed by the software designers to create the software. Consistent â€" The SRS should be consistent within itself and consistent to its reference documents. Ranked for Importance â€" 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. Traceable â€" 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? Very often we find that companies do not understand the difference between a System specification and a Software specification. What is the difference between a System Specification and a Software Specification? Very often we find that companies do not understand the difference between a System specification and a Software Specification. Important issues are not defined up front and Mechanical, Electronic and Software designers do not really know what their requirements are. The following is a high level list of requirements that should be addressed in a System Specification: In many systems we work on, some functionality is performed in hardware and some in software. It is the job of the System specification to define the full functionality and like the performance requirements, to set in motion the trade-offs and preliminary design studies to allocate these functions to the different disciplines mechanical, electrical, software. Another function of the System specification is to specify performance. Some portion of that repeatability specification will belong to the mechanical hardware, some to the servo amplifier and electronics and some to the software. It is the job of the System specification to provide that requirement and to set in motion the partitioning between mechanical hardware, electronics, and software. Very often the System specification will leave this partitioning until later when you learn more about the system and certain factors are traded off For example, if we do this in software we would need to run the processor clock at 40 mHz. However, if we did this function in hardware, we could run the processor clock at 12 mHz. I think it is useful to say that explicitly. This is done primarily because most of the complexity is in the software. When the hardware is used to meet a functional requirement, it often is something that the software wants to be well documented. Very often, the software is called upon to meet the system requirement with the hardware you have. Very often, there is not a systems department to drive the project and the software engineers become the systems engineers. For small projects, this is workable even if not ideal. In this case, the specification should make clear which requirements are software, which are hardware, and which are mechanical. In short, the SRS should not include any design requirements. However, this is a difficult discipline. For example, because of the partitioning and the particular RTOS you are using, and the particular hardware you are using, you may require that no task use more than 1 ms of processing prior to releasing control back to the RTOS. Although that may be a true requirement and it involves software and should be tested â€" it is truly a design requirement and should be included in the Software Design Document or in the Source code. Consider the target audience for each specification to identify what goes into what documents. It should define everything Software needs to develop the software. Thus, the SRS should define everything explicitly or preferably by reference that software needs to develop the software. References should include the version number of the target document. Also, consider using master document tools which allow you to include other documents and easily access the full requirements. Take your time with complicated requirements. Vagueness in those areas will come back to bite you later. This is a great question. There is no question that there is balance in this process. We have also seen customers kill good products by spending too much time specifying it. However, the bigger problem is at the other end of the spectrum. We have found that taking the time up front pays

dividends down stream. Here are some of our guidelines: Spend time specifying and documenting well software that you plan to keep. Keep documentation to a minimum when the software will only be used for a short time or has a limited number of users. Have separate individuals write the specifications not the individual who will write the code. The person to write the specification should have good communication skills. Pretty diagrams can help but often tables and charts are easier to maintain and can communicate the same requirements. Conversely, watch out for over-documenting those functions that are well understood by many people but for which you can create some great requirements. Keep the SRS up to date as you make changes. Test the requirements document by using it as the basis for writing the test plan.

Chapter 7 : A Software Design Specification Template

Design documents are also referred to as functional specifications or functional specifications documents (FSDs), or functional requirements specifications. What is a high-level design document? A high-level design document (HLDD) describes the architecture used in the development of a particular software product.

Chapter 8 : Design Specifications (DS) | Ofni Systems

The Software requirements specifications document might have implications about the design of the user interface, but these implications are typically superceded by a formal design specification and/or prototype.

Chapter 9 : Software Design Specification

Software requirements specification is a rigorous assessment of requirements before the more specific system design stages, and its goal is to reduce later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules. [1].