COM310 - Functional Requirements Document

Dustin Ingram

November 17, 2009

1 Introduction

This requirements document will serve as an official analysis and statement of Functional Design Requirements, specifically focusing on the design, development, deployment, and evaluation of a computer or software system. This system is to be highly utilized by workers who need to focus directly on numerical information, the analysis of it, and the creation of knowledge from it. It will act as a guideline to developers to determine the exact requirements of the system, and the design necessary to achieve a successful working environment. Specifically, as a Functional Requirements Document, it will primarily focus on what the system should do, including interacting with the user, rather than how it should be done.

1.1 Intended audience

The intended audience for this document is both a non-technical group and a technical group. The document must be able to be interpreted and understood by a non-technical group, such as upper management, as well as a technical group, such as the designers of the system, or a mix of technical and non-technical, such as the users of the system. For example, the designers of the system have highly technical knowledge of the actual requirements of the system - the "how" which describes how the system will function at the most basic level. However, this is not required knowledge for the intended user, and is likely not required knowledge for the manager or supervisor who will have to review and approve of the final

design.

However, since the document will be a combination of input from all three of these very generalized categories of "designers" of the system, it must contain a relevant mix of technical knowledge–too much, or too little, and the document may become useless for some users.

2 Gathering Functional Design Requirements

In his text, Koomey outlines a series of steps first presented by Hughes that a critical thinker will take to create a sound argument (Koomey, 59). Here, one can apply a variation of these same steps to the development of the Functional Requirements document, since ultimately, one's Functional Requirements must be a logically strong and sound argument, as Koomey and Hughes both affirm. The following key steps below are an adaptation of these steps to suit a Gathering of Functional Requirements.

- *Identifying the User's Main Conclusion*: It is important to examine the specific context of the situation for which the system is being designed, from the user's perspective. The problem must be clear and understood by all.

- *Identifying the Premise of the Problem*: At this point, one must outline the assumptions and needs for the user. Important care must be taken to reject erroneous or irrelevant premises, and to discover and include premises which might be unstated or taken for granted.

- *Identifying the Structure of the Argument*: When beginning to design the argument, it can often be helpful to simply outline, as best as is possibly, the underlying structure of the argument. For example, determining which premises are most important, which rely on each other, which stand independently, etc. If an argument contains a sub-argument, or the problem has a subset of problems within itself, these smaller elements must be tackled first, and then the entire design and structure can better be determined.

- *Justification of the Premises*: Often, premises can be made that are not relevant or even erroneous to the argument, as mentioned in the previous step, "Identifying the Premise of the Problem." A crucial part of eliminating premises that are wrong is providing a complete justification of the premises. As Koomey quotes, Hughes suggests two criteria for evaluating the justification of a premise, as follows:
  - "If the statement is common knowledge, we should regard it as acceptable, unless the context requires a higher standard of proof"
  - "If the statement is not common knowledge, we should ask for, or be prepared to offer, the evidence upon which it is based, and accept it only if the evidence meets the appropriate standard..."

  These two criteria form a sound method of evaluation for the justification of the premises of a series of Functional Requirements.

- *Relevancy of the Premises*: In the same vein as the aforementioned justification of the premises, the evaluation of the relevancy of the premises as well is indicative of good Functional Requirements gathering. As Koomey states, often there are justifiable premises which seem to support the ultimate conclusion, but are in fact irrelevant. Although Koomey suggests that previous steps should eliminate the need for this additional step, it can still be considered a required part of the process.

- *Support of the Conclusion by the Premises*: Again, besides proving that the premises are justifiable and relevant, they must also of course support the ultimate conclusion. While it is possible for a premise to be justifiable, relevant, but not support the conclusion, hopefully such a premise would be eliminated in previous steps–perhaps they are not sufficiently explained to make the argument they represent as adequate as it needs to be.

- *Missing Arguments in the Premises*: Often, this can be the hardest part of creating a

valid, comprehensive group of premises to support a series of Functional Requirements. Although it's understandably nearly impossible to see into the future or read a user's mind, sometimes the creation of a series of "possibly missing arguments" would help find any premises that are lacking in the current draft of the Functional Requirements. If this is the case, however, one must be sure to repeat previous steps to verify that any additionally created premises are justifiable, relevant, and ultimately support the conclusion.

- *Evaluating Counter Arguments*: It can be at times hard to see counter arguments when one is invested or has begun to examine what one thinks is the correct argument. The creation of such a contradictory argument (or the attempt to do so) may reveal some missing arguments in the premises, or simply cement the already existing argument. Koomey points out that if a sound (meaning justifiable and relevant) counter-argument is found, "something's wrong, because two sound arguments cannot contradict each other (unless one is not truly sound)."

In the same way that the creator of a series of Functional Requirements must verify the initial premises of the document, the creator also must take steps to verify that the user they are gathering the Functional Requirements from is a valid source of information, and will provide accurate and relevant results. Without a valid user-figure, it will be nearly impossible to develop premises that sold the problem, or create a valid problem in the first place.

One can adapt the steps that Koomey outlines to question an authority into steps to identify whether a user is a valid and true user (Koomey, 69). This is ideal, because ultimately, the user or users who are the source of the Functional Requirements and it's related premises should be an "authority" in the area that the system is being designed for. Therefore, one can see that these following steps, adapted from the Koomey text, can be absolutely vital.

- *Identify the Authority-User*: This step is perhaps immediately relevant: the importance

of accurately identifying the user is discussed above, however, besides simply finding the user, it is necessary to truly identify the user–to gather additional information. These additional sources of knowledge may give a better perspective and scope for the user's input and the way the user will interact with the system.

- *Recognizability of the Authority-User*: At a bare minimum, the user must be one of the key or primary users of the system for which it is being designed. The more closely involved the user will be with the system, and the intimacy they have with whatever existing system they may use, is the next logical step. The user must be recognizable to both the designers of the system, and the other users of the system.

- *Accountability of the Authority-User*: The user who is the basis for the document must be able to be held accountable for the input they give. If the user gives invalid data or feedback, this should result in a improperly designed system for that same user. This is often described as the "dogfood" philosophy–that the designer of a system, if relevant, should be able and willing to use the system they created. In this case, it means that the Authority-User must be directly affected by the system, and actually use it.

- *Credibility and Support for the Authority-User*: Along the same lines as the recognizability of the Authority-User, other users of the system must be able to identify that the user is credible, and be able to support the ideas and input that that user will give as their own.

These four steps give a firm foundation for the identification and selection of an appropriate Authority-User.

3 Functional Requirements

What follows is an identification of the important characteristics that Koomey outlines to develop the best possible analysis. In this case, one will develop these into a set of guidelines to design and support knowledge workers through the effective design of a series of Functional Requirements and the resulting system.

Koomey's system is well suited to be applied to this investigation of Functional Requirements in a Problem Solving Environment. Koomey has adapted his list from the author Peter Schwartz' book, *The Art of the Long View*, as a way "to explore several possible futures in a systematic way." Below will be a rationale or justification for each of the steps, as well as the important functional characteristics for the requirement.

3.1 Defining the Problem the System Will Solve

Koomey says, "Scenarios should be developed in advance of some major strategic decision so that when the time to decide arrives, the options are laid out in detail and the implications of each decision path are made manifest." In short, what Koomey means by this is that the absolute foundation for the system is an accurate description of the problem that the system will solve. This means that a large percentage of the actual development of the requirements should be spent making sure that the problem is accurate and will truly solve issues or improve the user's experience.

For the user, this comes down to the designer correctly interpreting the problem so that when the system is designed and completed, the problem is actually solved. Too often, the true issue is miscommunicated, misinterpreted, or otherwise obscured so much that the resulting system is far from solving the user's initial issue, amounting to an immense waste of time.

3.2 Determining the Factors Current Affecting the System

In the same way that the designers must validate and expertly define the problem that the system will solve, they must also outline the key factors which will decide whether the solution will, in fact, be a solution. Specifically, the designer must pay careful attention to find *all* factors affecting the system. Any factors omitted may change the success of the system and may render the solution impossible.

Functionally, for the user, this is the inclusion of all affecting factors in the user's working environment which affect the system. This ensures that no matter what factors may change or occasionally exist or not exist, the system will remain resilient and functional for the user. Here, one can see the importance of the user's communication to the designer about all the involved factors–if any are omitted, it is not guaranteed that they will be addressed.

3.3 Evaluating the Affecting Factors by Importance and Uncertainty

Correctly ranking and evaluating the affecting factors ensures that the correct amount of time is spent designing for each of the individually determined factors. Koomey writes, "...A small number [of trends and driving forces] will be both highly important and highly uncertain... These are the issues upon which the differences in the scenarios will be based. Predetermined elements don't differ among scenarios." His point is quite valid–when the designer is outlining possible scenarios to examine the system by, the importance of some factors over others is fundamental.

The functionality of the importance of affecting factors for the users is immediately apparent. Similarly to the case where factors are omitted, or irrelevant factors are not omitted, factors which are incorrectly ranked in importance create a system that is not as functional for the user as they should be. If the scenarios which are examined and evaluated are not relative to the actual scenarios that the user creates by interacting with the system,

the system will be inefficient.

3.4 Creating the System's Scenario Set

In the previous section, the evaluation of factors for scenarios was discussed. Here, one can see the importance of creating an entire set of scenarios based on relevant and important factors–if the designer has nothing to design against, the system will never be correctly "tuned." Koomey outlines the sub-steps for creating a scenario or scenario set as follows: "The most important and uncertain factors should be combined in various ways to define the set of scenarios to be considered." The correct combination of factors will result in the best of all possible sets of scenarios. Koomey goes on to say that "the set of scenarios chose also frames the story lines that will emerge from the scenarios." As mentioned before, these will provide the designer with the appropriate environments to prove the system.

From the user's point of view, this is to ensure that the system will be able to cope with a wide range of scenarios and events which they may "throw" at it. If the system is not able to react appropriately to the user, it will fail, and the usability of the system will fail as well. Choosing a wide and diverse, yet important scenario set will allow for a truly robust system to be designed.

3.5 Developing the System's Scenario Set

Not only should the Scenario Set be created, as is outlined in the previous section, but it should be developed into a full and complete set. This means adding specific details to the scenario to prove that they are not simple generalizations of common tasks, and that they are directly transcended from actual use cases. As Koomey says, "Each of the key driving forces and factors from [the previous step] should be assessed for each of the main scenarios." This means that for each of the scenarios created in the Scenario Set, they must be "peppered" with relevant factors from the previously determined list of factors, based on their importance as well.

Again, like mentioned before, for the user this means that the scenarios are not just generalizations but actual scenarios that they have encountered, or expect to encounter in the future when using the system in their work environment. It would be too easy for a system which has been designed with generalized scenarios as it's basis to fail to meet the needs of the user.

3.6 Determining the Implications of the System

Now that one has a firm grasp on the problem at hand, as well as the scenarios that will result, the next step will to be determine the ultimate implications that the designed system will have on the existing work process of the user. From the text: "Assess how [a] decision would fare in each of the different scenarios. If the decision would lead to success for all or most of the scenarios considered, that decision is probably a wise one."

At the most basic level, this is the effect the system will have for the user: whether it will increase productivity, create more usability, etc. Special attention should be made to determine whether the resulting implications are positive or negative, more positive than negative, or vice versa. While it's impossible to design a "perfect" system, the implications should be almost always entirely good for the user and their functionality. The implications the system will have on the user and their work environment will directly determine the success of the system.

3.7 Defining the Success of the System

Finally, the most important step is to determine whether the system is successful or not in it's final state, based on some designed metrics for success. For example, "...it is crucial to identify a few key indicators by which decision makers can monitor the evolution of events." Even if the system has been designed correctly with all of the previous steps, it is still possible for error to exist, or for the system to adapt over time. There must be a way for the designers to be able to realize when the system has or will fail in some way, and translate

that into eventual increased functionality for the user.

This is the most important step for the user. It acts as verification that the system will work based on their assumptions, the scenarios they have put forth, and the use cases they have experienced. Not only does it make sure the system works "out of the box", but that it allows the system to be adapted over time if scenarios were to change in such a way that the system becomes unusable or less functional. An important note from the text says " If you can't measure it, you can't manage it."

4 Conclusion

These outlined steps, which document the preparation for and design of a series of Functional Requirements, serve as a foundation to focus on the design, development, deployment and evaluation of a computer software system. If each of these steps are correctly followed, and the design process is as uninhibited as it can be, the resulting system will be a usable, productive and problem-solving system for the user, even at the most functional level.