Moti Begna
CSCI 5801
3/26/2019

SRS Reflection

1. **Elicitation Session**
   a. **What did you learn from the elicitation session? What parts worked well? What parts did not work well?**

   The elicitation session displayed a clear example of using an unstructured approach, whereby we constructed a list of varying questions not subject to any categorical assignment and proceeded to conduct the session by randomly selecting any of the questions we wanted to ask. In some cases, this approach worked well in that many varying questions were able to be asked so that we covered a wide understanding of how the system as a whole would work. In addition, some teams asked very similar questions, but received more clarification after the first instance that the question was asked. This also created an issue, however, seeing as how in some cases, clarifications weren't given but rather completely different answers that made the initial answer unclear and ambiguous.

   b. **How might you prepare differently for a similar experience in the future, where you may only have one chance to meet and talk with the user(s)?**

   In the future, if I was to only have one chance to meet and talk with a user, I would utilize a more structured approach to the elicitation session in which I would come up with a list of questions which were grouped in a way that made categorical since in relation to the various requirements of the system. I would also formulate a framework for the session itself in which all questions of a given category where asked before moving on to the next category. Overall, I would much rather prefer that any future elicitation sessions were more organized and prepared for in order to not only get better answers from the users, but to also present my team/myself as being organized and thorough.

   c. **When is asking the same question twice bad/unhelpful? When is asking the same question twice good/helpful?**

   As mentioned before, it can be very helpful to ask the same question twice if the answer given the first time was unclear and/or needed more elaboration. This ambiguity often occurs since sometimes it's necessary to move on to other questions that may spark a better answer from the user when the question is asked a second time. This approach can sometimes be harmful however if the answer given the second time around is vastly different than the first answer. At this point, the user might be considered an unreliable source for answers related to that question.

2. **Individual Requirements**
   a. **How well did capturing the requirements go?**

   Because we were given a basic framework of what the system should look for the various users that would handle it, it wasn't to difficulty to word/elaborate on the functionalities of the system in order to transpose it into a requirement. Organization was simple as well, given that the various user entities provided the different sections for the requirements.

For example, Prospective TA's had their own section of requirements that directly pertained to how they would be able to use the system. One of the more difficult aspects of capturing the requirements was reviewing the statements given in the framework that were written in a way that could be left to interpretation.

b. How did you approach user statements/requirements which were vague or ambiguous?

The biggest issue when it came to user statements which were vague and ambiguous was transposing them into requirements which both clarified the statement but also kept it general enough so that developers could have the option of implementing the requirement how they wanted. My approach to these kinds of statements was to use what I knew about how the system should work to best clarify the statement in a way that made since as a requirement. For example, users are able to submit their technical scores into the TA application, but what kind of scores they could enter were not given. Thus, I clarified the requirement by stating that users may optionally enter their TOEFL and SETTA scores since these scores relate to how TA's are appointed.

c. How did you approach user statements/requirements which were contradictory?

I didn't find any requirements that were contradictory, however my approach to settling conflicting requirements would be to eliminate the requirement that least fit what a user would want from the system. In the end, the requirements are meant to outline what the user would like, thus any contradictory requirements that vaguely/hardly met what the user wants should be removed. This also means that I would have to review any other requirements that depend on the contradictory requirement that I chose to remove, deciding to reword or also remove them.

d. Did you follow a template? Did the template help or hurt your efforts? Why?

For the most part, I utilized the IEEE template for my requirements. This template helped my efforts in that it gave me a basic understanding of how each requirement can be formulated such as basic descriptions, priority levels, related use case, and inputs/outputs. All in all, it gave me an understanding of what requirements can look like, allowing for variability on the part of the writer if they chose to add or omit certain segments.

3. Team Requirements

a. Did you encounter any situations where you and your teammates disagreed on the definitions or process? What caused that ambiguity? What could you do next time to avoid it? If your team faced no disagreements, explain how you worked together to build the requirements to avoid them.

For the most part, my team faced no disagreements with how the requirements were built and were in fact supportive of all of the ideas that we came up with. We decided to build upon the requirements written by one of our teammates since we unanimously agreed that their requirements were more detailed and better organized. After deciding this, we made sure that each teammate was in charge of editing/building upon one section of the requirements so that there were no conflicts during the process. It was then that we went over all of our work to review each of our additions, and then made changes accordingly whenever we agreed that changes should be made.

4. <u>**Final Spec & Testing**</u>

   a. **What kinds of assumptions will be necessary to ensure that meeting the specification (verification) means that you have also met the requirements (validation)?**

      In order to ensure that meeting the specification means meeting the requirements, we had to have a couple of assumptions about the system. Firstly, we have to assume that implementation of our system within the University's larger system would cause no issues in regard to any conventions used in the pre-existing system. This could be naming conventions, University system guidelines, and privacy concerns. In addition, we also have to assume that the system will be maintained after release so that if any errors or downtime occurs, there will be people to ensure that all of the errors have been fixed and the system is once again operational. Another assumption that must be made is that all users of the system will be knowledgeable enough to be able to use the system. If this isn't the case, we must assume that users will be taught in how the system works.

   b. **How did you translate the requirements to specification?**

      To translate the requirements to specification, we had to analyze what the system should first initialize for the requirement, what it should be doing while the requirement is in progress, and also what it should do after the requirement is met. When it comes to, for example, initializing the TA application, the system must create various listing options as well as text input modules for the various answers that users would give. During the process, the system must ensure that every required part of the application is being filled, and after the process, the system must submit the application into some defined database. Essentially, translating requirements to specifications means describing the background actions of the requirement that is meant to specify how the how the system would meet the requirement.

   c. **What was the most difficult aspect of writing your test cases?**

      The most difficult aspect of writing the test cases was ensuring that we covered all of the important scenarios for each requirement. This meant that we had to build test cases for not only when the system works as it should, but also for when errors could possibly occur. For example, while a user can appoint a prospective TA into a position, it should be noted whenever they try to appoint someone that has already been appointed before by either them or another user. Formulating this test case took thinking about not only what the user could do but also the possible results of their actions.

   d. **Did writing your test cases help you find issues in your requirements? If so, what did you find and how did test case writing help you find/fix it? If not, what kinds of work effort (man-hours/financial) is going to be necessary to implement your tests? Explain.**

      During the process of writing our tests, we did come across missing information regarding how we would house all our TA applications as well as where we would keep the names of all the appointed TA's. This occurred when we were writing tests for when users needed to access information regarding applications and appointed TA's. Specifically, it was when we had to think about what the system should be initializing so that users could access the information that we realized that we needed a system database to hold the information. Writing the test case helped us by finding new angles to how each of the requirements should be met which we would have otherwise not thought of.

e. **How brittle are your tests? What kinds of changes could break them (provide concrete examples)?**

   For the most part, we kept our tests general enough so that any changes to our requirements later down the line wouldn't drastically impact the tests as a whole. However, some of our tests would cause false-negatives if our requirements were to change. For example, as it stands users must login using their x500 and password in order to use the system based on their access level. Otherwise, the system would not give them access. If we changed how users could login in later, however, then our test case designed to not give access to users if they did not provide an x500 and password would trigger an unauthorized response, even though it should give them access. Another change that could break some of our tests is if we ever decide that appointing TA's would be a multi-step process in which multiple administrations would have to sign off on a prospective student before appointing them into a TA position. Currently, our tests simply ensure that when an administrator chooses a TA, they are directly sent into our Appointed TA's database. However, if more verification is needed, then new tests must be made in order to accommodate for this change.

5. **Misc.**

   a. **What were the most difficult parts of writing the SRS (including Assignments 1-3 in your thoughts)?**

   One of the more difficult aspects of writing the SRS was ensuring that we captured as many major use cases as we could possibly think of. While a majority of the use cases were written when we first wrote the user requirements, it wasn't until we got to other aspects of the SRS such as the UML diagram and test cases that we realized the our use cases either didn't entirely capture what we initially thought we needed or that we needed multiple new use cases to capture areas of the system that we missed. In addition, it was also difficult to create the interaction diagrams since most of us were unclear if we were detailed enough in our drawings of the diagrams. We also ran into this same issue of descriptiveness in our test cases.

   b. **What additional experience or training would be helpful for next time?**

   I believe that if we went over interaction diagrams and test cases in class a bit more, these specific sections would have been less difficult to formulate. While enough information was given to get a basic understanding of these sections, I think it would be helpful if we went over more examples that more closely tied into the system that we are outlining.

   c. **How sure are you that the specification you wrote meets the requirements? If you are confident that it does, what specifically gives you that confidence? If you are not confident, what would you do to fix that, and how do you know when to stop?**

   For the most part, I am fairly confident that the specifications we wrote met the requirements that we originally outlined. Specifically, I feel this way since I believe that we thoroughly thought through what the system should be doing before, during, and after a requirement is met. This thinking was guided by our creation of our tests since we had to think about those three situations for every requirement. In addition, we kept our specifications general enough so that modifications to a part of the system could be made later in the implementation process based on any changes that users may want to incorporate later down the line.

d. What are the greatest risks in your SRS? What are the likely things that will go wrong? What makes them risky?

Because we made our SRS fairly general, there aren't many risks to it that could make it go wrong. However, as mentioned before, if we were to change how users are given access into the system, this change would cascade through not only our requirements, but also our use cases, UML diagram, and our test cases, requiring us to making many adjustments to the SRS as a whole. Our generality of the SRS could also be an issue if a requirement was to be interpreted in a way that would, rather than meet the users requests, cause additions and/or changes to how the system operates that it otherwise shouldn't be causing.