THE MODIFICATION PROCESS: A PRACTICAL MEANS TO UNDERSTAND AND ENHANCE THE SOFTWARE REQUIREMENTS ENGINEERING PROCESS

Julie Hatalsky
Trimble Navigation, Ltd.
9825 Huer Huero Road
Creston, CA 93432 USA
(805) 438-3418
julie_hatalsky@trimble.com

Paul S. Corwin
Clark Savage Turner, J.D., Ph.D.
Department of Computer Science
Cal Poly State University
San Luis Obispo, CA. 93407 USA
(805) 756-6133
pcorwin@calpoly.edu, csturner@calpoly.edu
www.csc.calpoly.edu/~csturner

ABSTRACT

Software requirements elicitation is a difficult process with many existing problems, and no single elicitation method solves all these problems. We introduce a new way of looking at the requirements elicitation process. Our model shows the requirements elicitation problem as a process of merging the users' wants and the users' needs into the same entity. In the context of our model, existing processes can be enhanced to help ensure that the users' wants and needs are both met. In doing so, we show that resulting requirements are more correct, complete, and feasible. Furthermore, using our model can help limit the solution to what the user truly needs, thus reducing unnecessary complexity.

KEY WORDS: Software Requirements, Software Engineering, Software Methodologies, Prototyping

1. INTRODUCTION

Software requirements engineering is hard. Requirements must be complete, correct, feasible, unambiguous, consistent, readable, testable, and traceable [1]. Trying to create requirements that satisfy all of these characteristics is a difficult and complex task. According to Brooks, [2] no technique or combination of techniques will ever significantly reduce this complexity; however, one can manage complexity by designing only the software that the user wants and/or needs. But which should we, as developers, provide them with?

An ongoing debate in software engineering is whether the goal of a software project should be to give the customer what they *want* or to give the customer what they *need*. Beizer [3], in enumerating a list of questions to which 'yes' should be answered regarding a software package, asks: "Does it have the features the users need (as contrasted to want)?" By making note of need being contrasted to want, Beizer implies that the two are not one and the same, and that it is more important to provide the users with what they need. On the contrary, in Gause and Weinberg's classic text on requirements [4], they go so far as to say that it is a mistake to try to give customers what they need rather than what they want. This paper investigates the issue further to envision the ideal where the software requirements process could merge what the customer wants and what the customer needs into the same idea. We take the problem from a high level discussion and dissect it to understand in a deeper and more practical manner what happens in the requirements engineering process.

Taking a closer look at [4] suggests another way to look at the problem. They state, "If you find yourself feeling that you know better what the customers need, … try to convince your customers that they really need what you think they need. If you can't convince them, either produce what they want, or find yourself some other customer." This suggests that the customers' ideas of the project consist of the *wants* and the developers' ideas of the project consist of the *needs*. That is, at the start of the project the customers have an idea of what they want, the User Set of solutions. The developers have an idea of what the customer needs, the Developer Set of solutions. At the end of the requirements engineering process, these two sets should intersect such that a common solution can be created that exists within both sets. In this paper we discuss not only the User and Developer Sets, but we also introduce a third set, the Constraints Set.

In this paper, first we introduce and discuss the Three Sets of software requirements. Then we discuss how the sets are modified during the requirements elicitation process. Next we discuss the possible outcomes of a simple requirements process viewed in the context of our model. We look at the ways in which the shape of the sets can end up and what each relationship means. Then we give

an example of the model in use followed by conclusions and suggestions for future work.

2. THE THREE SETS OF SOFTWARE REQUIREMENTS

User and Developer Sets

At the beginning of a software project the users have an idea of what they *want* implemented. We define this as the User Set of solutions. The make-up of this set changes constantly throughout the software creation process and, in fact, many the initial members of the set would not satisfy the users' needs [4]. In addition, the developer has an initial idea of what the users *need*. We define this as the Developer Set of solutions. This set is also constantly changing and the initial set probably has few members that would actually satisfy the users' needs [2]. The two sets initially may have very little or no overlap (see Figure 1).

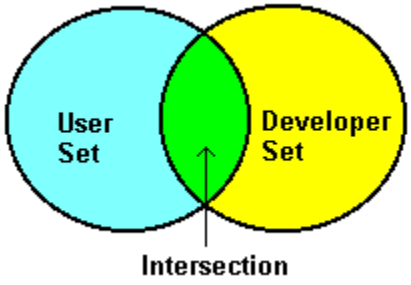

Figure 1: User and Developer Solution Sets

These sets can be looked at as the *perceived* wants of the user (User Set) and the developers' *perception* of the users needs (Developer Set). Throughout the requirements elicitation process, these sets change to more accurately mirror the set of solutions that would actually satisfy the user. These are the *actual* needs as opposed to the perceived wants and needs. We discuss the movement of the sets more thoroughly in Section Three.

Taking a solution from the intersection of the two sets helps insure that the actual needs of the user are met, adding to the completeness and correctness of the requirements.

The Constraints Set

Thus far we have discussed only the users' and the developers' ideas of what solutions will satisfy the users' needs. However, there is an important third set, the Constraints Set, that affects the software engineering process. "Constraints are restrictions that are placed on the choices available to the developer for design and construction of the software product." [1] This set contains solutions that satisfy certain project constraints, such as schedule, cost, complexity, staffing, software speed, and necessary hardware, to name only a few of the many possible constraints.

As noted above, taking a solution from the intersection of the User and Developer sets helps insure that the users' wants and needs are met. Also notice a key difference between the Constraints Set and the User and Developer Sets: a satisfactory solution *cannot* be taken from outside the Constraints Set. The Constraints Set represents all of the solutions that would satisfy the users *actual* needs (as opposed to perceived wants and needs) that can be achieved given the current stated constraints. It would be infeasible to build a satisfactory solution not taken from this set, in terms of cost, man-hours, or some other possible constraint. To insure the feasibility of the project as well as the users' acceptance of the eventual product, the solution must be taken from the intersection of all three sets (see Figure 2).

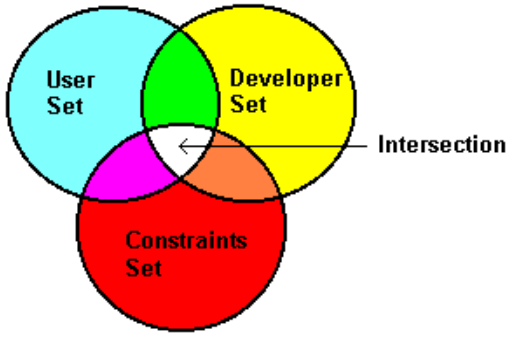

Figure 2: Three Sets of Software Requirements

Breaking Down the Sets

The idea of one User Set, one Developer Set, and one Constraint Set is an over-simplification used to help visualize the problem. To help our understanding of the requirements engineering process we need to look at each of the Three Sets as the representation of many smaller sets. For example, there typically is not one user involved in a software engineering project. There can be many different individuals and even different types of users involved [1]. For example, the following different sets could be contained in the User Set for a specific project:

- User (Project Champion)
- Actual End User
- Project Executive Sponser
- Project Marketing

The intersection of these smaller sets is the actual User Set (see Figure 3).

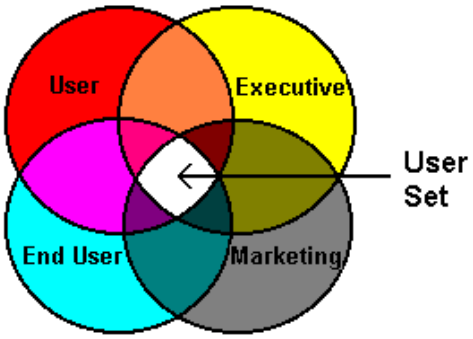

Figure 3: Possible Make-up of the User Set

The Developer and Constraints Sets can be similarly broken down.

3. THE MODIFICATION PROCESS

The Modification Process is simply another way of looking what is actually happening during the requirements engineering process. We call this process the Modification Process because it *modifies* the Three Sets described above. Set modification can either expand or shrink a set through the requirements elicitation process. This is done by eliminating possible solutions from a set or by adding new possible solutions to a set. For example, consider a software system that keeps track of member data for a gym. The user informing the developer that the software must be able to keep track of how long a person has been a member changes the possible solutions in the Developer Set. The effect can make a set appear to be moving. The overall effect of the process should make the three sets merge into one Intersection Set (see Figure 4) that the eventual solution is taken from.

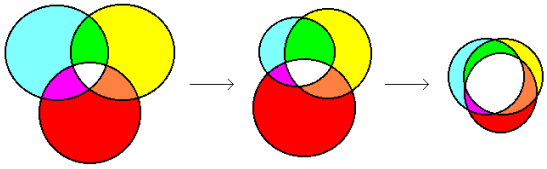

Figure 4: The Modification Process through requirements elicitation

The idea of the sets moving does not necessarily indicate that the problem itself is changing. While the problem generally remains static, the perceived sets shift as information is revealed and discoveries occur [1]. An exception to this is the Constraints Set whose makeup is initially the solutions that will actually satisfy the users' needs given the current stated constraints. The User and Developer Sets shift during requirements elicitation with the goal of more accurately matching some actual solutions contained in the Constraints Set. However, the Constraints Set can also change during the Modification Process. Some features of the Set, such as cost and man-hours, could be deliberately changed by the user. Other features might vary depending on factors not under the control of either the user or the developer, such as economics or business cycle.

Goals of the Modification Process

Earlier we claimed that our model would help make the resulting requirements more feasible, complete, and correct. In addition, building only what the customer actually wants would reduce complexity. With that in mind, our goal for the Modification Process, is to modify these sets such that:

- The Intersection Set exists

This insures the feasibility of a solution as the developers and users perceive the problem.

- The Developer Set lies solely within the Constraints Set

This insures the correctness and completeness of the requirements. The solutions that the developer might build will all lie within the set of actual solutions.

- The Developer Set covers a maximal portion of the Constraints Set

This gives the developers the most flexibility to choose a solution.

- The intersection of the User and Developer Set is as large as possible

This insures that the solution the developers build will satisfy only what the customer wants, reducing unnecessary complexity.

4. OUTCOMES OF THE MODIFICATION PROCESS

There are two possible outcomes at the conclusion of the Modification Process:

- The Three Sets intersect
- The Three Sets do not intersect

In other words, the Intersection Set will either contain at least one solution or it will be empty.

The Intersection Set Exists

If the Intersection Set contains solutions, the developers can continue the software engineering process and begin creating a solution. Of course, the Sets could continue to change during the software creation process, causing the Intersection Set to shift [1, 4, 5, 6]. This could cause the

chosen solution to fall outside the Intersection Set, necessitating a change in solutions.

The developers are only aware of the solutions that exist in the Developers Set, as these are the solutions that they see to the problem. The solution that the developers choose to build can come from four possible sections of the Developer Set (see Figure 5). The following figure and corresponding section explanations show why it is important that the Developer Set lie as much as possible inside the Constraints Set.

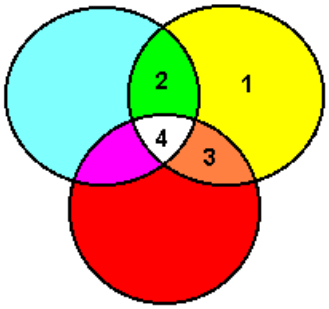

Figure 5: Developer Solution Choices

1. A solution taken from this set will not be successful. It either does not satisfy the users' actual needs or cannot be built given the current constraints.
2. A solution taken from this set will also not be successful, at least initially. It may satisfy all of the actual needs of the user, but cannot be built given the current constraints. If the users and developers agree that a solution from this set is the best one to take, then the constraints must be loosened so that the solution falls within the new Constraints Set.
3. A solution taken from this set may be successful. The final solution will satisfy all of the users actual needs. However, the developers will not have the support of the users during the construction process. The users will believe that the end product is something that they do not want.
4. A solution taken from this set will be successful. It will satisfy all of the users' needs and can be built given the current constraints.

The Intersection Set Does Not Exist

If the Intersection Set is empty, then there is no solution to the problem as it is currently understood. To obtain a solution one or more of the sets needs to be shifted. The next steps taken in the requirements process depend on which sets do not intersect.

The situation in which the Constraint Set does not intersect with the other two sets is shown in Figure 6. This is the situation where the users and developers have a common understanding of the problem, but the problem can not be completed given the stated constraints. For example, the solutions that the developers and users agree on might not be feasible considering the budget or man-hours assigned to the problem. To create an intersection of the three sets, the users can modify the constraints on our example project by agreeing to a larger budget or longer projected finish date, thus moving the Constraints Set [7]; otherwise, the users can modify their wants. For example, some less important features could be left off the project, thus moving the User and Developer Sets. If the Constraints cannot be further stretched and if only the necessary features are left on the project, then the project should be abandoned.

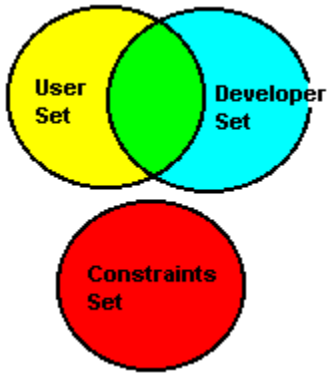

Figure 6: Disjoint Constraints Set

Another possibility is that the User and Developer Sets do not intersect. This occurs when the users and developers can not come to a common understanding of the problem (Figure 7). This could be because the users and developers come from such widely different knowledge bases that they cannot understand each other. Or, as Gause and Weinberg suggest, "It's not a good idea to work for people whose intelligence is so disparate from yours, in one direction or the other." [4] Regardless of the reason, it might be wise for the developer step down and suggest that someone else continue the project.

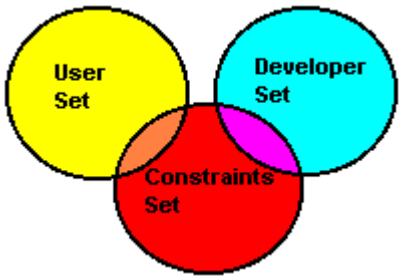

Figure 7: Disjoint User and Developer Sets

Another situation could be where the Developer and Constraints Set do not intersect (see Figure 8). This could be due to the developers not having the necessary skills or manpower to complete the project. Here, the developer could either step down or acquire the necessary training. Or, in the latter case, more manpower could be allocated. Acquiring the necessary training and allocating more manpower might necessitate a changing of the Constraints Set by allowing more time to finish the project and/or more cost for development [1, 8].

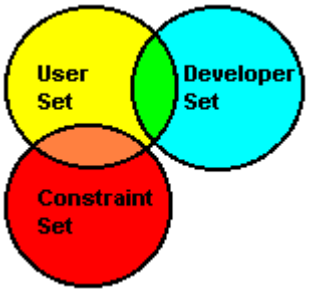

Figure 8: Disjoint Developer and Constraint Sets

5. USING THE MODIFICATION PROCESS TO ENHANCE EXISTING PROCESSES

The Modification Process is not limited to any one form of requirements elicitation method. It is a model that gives insight into what actually happens during any requirements elicitation process, whether it be a form of specification, rapid prototyping, or some other process. By looking at their own requirements elicitation method in the context of the Modification Process, developers can shift their focus and enhance their method.

Consider, for example, a generic situation where there exists a problem that a user calls upon a developer to solve. It is not uncommon at the start of a project for the User and Developer Sets to be placed inaccurately. That is, they are likely to contain false solutions and inaccurate assumption. Furthermore, real solutions may lie outside the Sets due to overlooked facts. Gauss and Weinberg [4] note, "We, as normal human beings, are just not very good at seeing what we've overlooked." For example, developers working on creating a requirements document for the A-7 aircraft [9] state that producing a list of fundamental assumptions forced them to also list some implicit assumptions. One reason for success of this process is that the reviewers of the document had a much easier time recognizing errors than they did recognizing omissions. Developers can therefore improve the requirements by explicitly concentrating on shifting the sets so that these sets become more accurately placed.

To more accurately place the User and Developer Sets, the developer might make a list of his own assumptions and interactively review them with the user. The user can then point out false assumptions to the developer, resulting in a shift or resize of the Developer Set. Likewise, the user can be made aware of ideas that he had overlooked, thus altering the User Set as well. In both cases the outcomes contribute to the clarification of the problem, which in turn leads to a state of elevated correctness and completeness in the requirements.

6. CONCLUSIONS

The Modification Process model can enhance the requirements elicitation process by:

1. Increasing the feasibility of a solution as the developers and users begin to perceive the problem.
2. Insuring the correctness and completeness of the requirements. The solution that the developer chooses to build will have a better chance of satisfying the user.
3. Giving the developers the most flexibility to choose a solution.
4. Insuring that the solution the developers build will satisfy only what the customer wants, reducing unnecessary complexity.

Developers can enhance their own process by reviewing and updating their current requirements elicitation method within the context of our Modification Process.

7. FUTURE WORK

Future work by the authors will include research into which existing requirements methods fit most naturally into the Modification Process. A prototyping methodology might best achieve the goals listed in Section Three. A comparison of different requirements methods in the context of the Modification Process would be interesting.

REFERENCES

[1] Wiegers, *Software Requirements*, Microsoft Press, Redmond, Washington, 1999

[2] F.P. Brooks, *No Silver Bullet: Essence and Accidents of Software Engineering*, Computer, Vol. 20, No. 4 (April 1987) pp. 10-19.

[3] B. Beizer, *Software Is Different*, In Proceedings Quality Week Conference, 1996.

[4] D.C. Gause, and G. M. Weinberg, *Exploring Requirements: Quality Before Design*, Dorset House Publishing, New York NY (1989).

[5] Boehm, *Software Risk Management: Principles and Practices*, IEEE Software, Vol. 8, No. 1, January 1991

[6] Gordon, Bieman, *Rapid Prototyping: Lessons Learned*, IEEE Software, January 1995

[7] T. Naslund, *Computers in Context – But in Which Context?,* Computers and Design in Context. MIT Press, 1997

[8] Hsia, Davis, Kung, *Status Report: Requirements Engineering*, IEEE Software, November 1993

[9] K.L. Heninger, *Specifying Software Requirements for Complex Systems: New Techniques and Their Application*, IEEE Transactions on Software Engineering, Vol. SE-6, No. 1 (January 1980) pp. 2-13.