

# **Software Lifecycle Using Visual Requirements Modeling**

A research paper submitted by

**Kunal Mittal**

**kunal@kunalmittal.com**

CSE8314 - Software Metrics and Quality Engineering

Spring, 2002

**Prof. Dennis J. Frailey**

## Table of Contents

<b>1. EXECUTIVE SUMMARY.....</b>	<b>3</b>
<b>2. OVERVIEW OF PAPER .....</b>	<b>5</b>
<b>3. WHAT IS THE PROBLEM?.....</b>	<b>6</b>
3.1.    QUALITY OF REQUIREMENTS .....	7
<b>4. REQUIREMENTS PROTOTYPING .....</b>	<b>9</b>
4.1.    TRADITIONAL SOFTWARE PROTOTYPING .....	9
4.2.    WHAT IS VISUAL REQUIREMENTS MODELING?.....	10
4.2.1. <i>Requirements Modeling Vs Traditional Prototyping</i> .....	10
4.2.2. <i>Motivation for Visual Requirements Modeling</i> .....	10
4.2.3. <i>B2IT Collaboration using Visual Requirements Modeling</i> .....	11
4.2.4. <i>Requirements Modeling Language</i> .....	12
<b>5. VISUAL MODELING AND TRADITIONAL SOFTWARE DEVELOPMENT MODELS.....</b>	<b>13</b>
5.1.    WATERFALL MODEL .....	13
5.2.    EXTREME PROGRAMMING .....	14
5.3.    RUP .....	15
<b>6. BENEFITS OF VISUAL REQUIREMENTS MODELING .....</b>	<b>17</b>
<b>7. REFERENCES.....</b>	<b>19</b>
<b>8. ANNOTATED BIBLIOGRAPHY.....</b>	<b>21</b>

## 1. Executive Summary

Research groups such as the Standish Group have shown that eighty four percent of all software projects fail, where Requirements are the number one reason for failure. The requirements phase traditionally takes forty percent of the SDLC. According to the “CHAOS Surveys and Reports” from the Standish Group ([www.standishgroup.com](http://www.standishgroup.com)), eighty-four percent of all software projects fail, where Requirements are the number one reason for failure. Statistics shows that only twenty six percent of all projects are successful. The same study shows that projects, on average, are over budget by ninety percent and take one hundred and twenty percent of the time as compared to the original estimates. Darrell describes how requirements related issues tend to be the primary cause of these overruns<sup>5</sup>. Thus ensuring that this phase is optimized and correct will increase the chances of success of software considerably. It has been shown that errors are cheaper to fix if detected early. Software errors due to inadequate or misunderstood requirements account for more than fifty percent of all errors in software<sup>8</sup>. It is clear that traditional techniques fail to catch many of these requirements errors. These studies clearly bring to light the need for a more optimized and complete Requirements process.

This paper explains the various traditional software development life cycles (SDLC) methodologies and the problems with the Requirements Phase in each of these methodologies. We demonstrate how we can use prototyping techniques, in conjunction with traditional requirements gathering tools and techniques, to addresses the issues with this critical phase of the SDLC. This is defined as a Requirements Definition Phase. Requirements Definition, not only addresses problems with traditional requirements gathering, but also addresses the issue of improving the quality of requirements using prototyping to test and validate requirements. This ensures a complete and accurate set of requirements. Furthermore, the development of the prototype and close interaction of business users and developers leads to B2IT collaboration, ensuring that the developers interpret the requirements correctly.

In order to justify a ROI for the introduction of a Requirements Definition Phase into the traditional SDLC, we need to define the Quality of Software. Traditional software quality can be described as its conformance to its requirements. However, we show you that this definition is not useful, as it does not address the issues of poor, missing, or ambiguous requirements. Thus, we define the quality of requirements. The quality of requirements can be defined as the conformance to the needs of the users or the business. A requirement is said to be of high quality if it is concise, complete, unambiguous, consistent, verifiable, modifiable, traceable, and understandable. Using this definition, the definition of quality of software can be defined as its conformance to its requirements.

## 2. Overview of Paper

Approach: EXPLORATORY – We expand on the work done on explaining the emphasis of Prototyping before development. We present new ideas, on how prototyping effects the software development life cycle (SDLC) and how it is into the SDLC in a cost effective manner. Our research presents ideas most applicable to the development of software for the Internet.

We examine the Requirements Gathering and prototyping processes, and show they add value to the traditional SDLC. The paper will emphasize how modeling requirements using visual and interactive prototyping can help validate existing requirements and identify missing requirements, before costly development dollars are spent. Quality is defined as a systems conformance to its requirements. Validating requirements can overall increase the quality of the system, by increasing its conformance to requirements.

We apply these concepts and present a SDLC model that introduces an iterative prototyping phase, before any design and development phases. This can be used to reduce the time for any iteration caused due to missing, incorrect, or incomplete requirements. We try to show how spending cycles iterating over the Requirements and Prototyping phases costs less, in terms on time and money, as compared to iterating over the entire lifecycle (Spiral, Waterfall models). We define the term requirements definition as the process of gathering, prototyping, testing and thus validating requirements.

We apply these concepts to other development models, such as the Rational Unified Process and Extreme Programming. Our research presents various models where the software development life cycle can benefit from requirements prototyping.

### 3. What is the problem?

Over the past fifty years, the organizational use of Information Technology (IT) has undergone a significant evolution. During this evolution, IT solutions have become increasingly complex and mainstream. IT is becoming a dominant and necessary part of any business. In response to this, various techniques and methodologies for developing and implementing software have been adopted.

The traditional model of software development relied on the assumption that business users and developers could collect and freeze the requirements<sup>2</sup>. According to the “CHAOS Surveys and Reports” from the Standish Group ([www.standishgroup.com](http://www.standishgroup.com)), eighty-four percent of all software projects fail, where Requirements are the number one reason for failure. The requirements phase traditionally takes forty percent of the SDLC. Requirements related issues were the main source of problems that plagued the software development process. Statistics shows that only twenty six percent of all projects are successful. The same study shows that projects, on average, are over budget by ninety percent and take one hundred and twenty percent of the time as compared to the original estimates. Darrell describes how requirements related issues tend to be the primary cause of these overruns<sup>5</sup>.

The issues with traditional Requirements Gathering phases are that

- Requirements are not testable – how do you know they are correct?
- No way to find the missing requirements
- No way to be sure developers interpret requirements properly

Requirements gathering is a rapidly changing process. RUP, XP, and other SDLC models do not address this process as a changing or iterative process. Traditional software development lifecycle’s such as the Waterfall model, Rational’s Unified Process (RUP), Extreme Programming (XP), and others do not concentrate on the Requirement Process.

They all show Requirements Gathering as a step in their process, but do not place any special importance to the step, or describe why it is probably the most critical step in the entire process. They relied on the assumption that end-users (stakeholders) and engineers could define the requirements accurately and correctly, and thus freeze some sort of a requirements documents. However, as seen with most software systems, initial requirements usually need to be revised after the first release of the software. Requirements are rarely even accurate before the user can see the system actually implementing the requirements.

### **3.1. Quality of Requirements**

Quality of software is the degree to which the application meets the requirements of the business. Generally reported in terms of defects against scope, quality represents the extent to which the web application is ready for release to users and the specific mapping of completed functions to the business requirements. However, this definition of quality can be elusive. Traditionally, the software quality is measured as its conformance to requirements. It is forgotten, that if the quality of requirements is poor, even if the software is of very high quality when compared to the requirements, it might not meet the needs of the business. The fault does not lie in the software or the quality of the software; rather it is in the quality of requirements. The industry needs a measure of requirements quality, and a process to ensure the quality of requirements is measured, maintained, and improved.

The quality of requirements can be defined as the conformance to the needs of the users or the business. A requirement is said to be of high quality if it is concise, complete, unambiguous, consistent, verifiable, modifiable, traceable, and understandable. If we use this definition for the quality of requirements, then we easily and correctly define the quality of software as the conformance to its requirements. Using this definition of quality, we can now apply all traditional software metrics to the measurement of quality of requirements. The best way to justify the new steps in various SDLC processes

followed for years is to show the benefits quantitatively. Thus, though measurement of requirements quality is not the focus of this paper, we will briefly introduce measurement ideas for measuring the quality of requirements.

## **4. Requirements Prototyping**

According to Andriole, prototyping is always a cost-effective and always improves specification<sup>3</sup>. The concept of prototyping a software before it is built is not new. Many organizations strongly believe in prototyping software before they invest development dollars, and other do not. Prototyping is a highly recommended approach, because review of the prototype enables users, investors, and engineers to agree on how an application should look and behave. In some cases, a prototype enables a software engineer to confirm that a particular requirement.

### ***4.1. Traditional Software Prototyping***

Requirements Prototyping is confused easily with the first release of the software. Some of the traditional SDLC processes talk about Requirements Prototyping. They talk about creation of a prototype of software before or simultaneously with the design phase. The code created during the prototype is expected to be re-used when the actual software, and thus the creation of the prototype is sometimes labeled as the beta release or at least included as a first release in the software roadmap.

Prototypes can be developed through a multitude of proven activities and tools. The basic inputs to these activities are the business requirements and functional specifications. The objective of application prototyping is to communicate functionality between IT and business stakeholders. This can often result in refinements made to the business requirements and specifications. Prototypes should consider technical constraints for the function being defined. Regardless of the format chosen for modeling or communicating the functional specifications of the application, it is important that the format chosen can be leveraged as the acceptance test criteria for the application. Using this approach, the duration of application development life cycle can be shortened and the application developed can be reconciled to its definition.

## **4.2. What is Visual Requirements Modeling?**

Requirements modeling is the conceptual model of the enterprise as seen by the end-users<sup>6</sup>. We introduce the concept of Visual Requirements Modeling as an alternative to software prototyping. We introduce this concept, primarily to combat some misconceptions of what a prototype needs to be. For purposes of introduction into the SDLC, we will call this as a Requirements Definition phase, which can be defined as the collection of accurate and complete requirements for software, by prototyping, testing, and validating each requirement.

### **4.2.1. Requirements Modeling Vs Traditional Prototyping**

A prototype should not be confused as a release of software, as explained in the previous section. The sole purpose of a prototype is to validate the requirements of software. It should be described as a process of creating a visual model of the software in a time and cost effective manner, in order to validate the requirements. Performance, scalability and other non-functional requirements need not be addressed by the prototype. Rather, it should concentrate on business processes, integration points, and the user experience.

### **4.2.2. Motivation for Visual Requirements Modeling**

Our concept of Visual Requirements Modeling is also targeted to resolve the three primary issues with traditional Requirements Gathering, such as testability, identification of missing requirements, and verification that developers have interpreted the requirements correctly. A visual system, allows the user to improve the requirements, identify missing requirements and thus overall improve the quality of the requirements. We are suggesting the need for some sort of a visual modeling tool for requirements verification, that will also users to collect requirements in a central location (using a Requirements tool such as Rational Requisite Pro), and then build a prototype based on the requirements, and linking each requirement to some step in the prototype. Being able to visually map the elements of the prototype to individual requirements, ensure that all

requirements are modeled, tested, and verified. Each requirement is linked to an element in the prototype, to ensure complete coverage and testing of all requirements. This helps find missing requirements, clarify ambiguous or incorrect requirements, and validate the quality of each requirement. This is easier to do in the prototyping phase, because the code is significantly simpler than production code and is expressed in a notation tailored to support modification<sup>2</sup>.

### **4.2.3. B2IT Collaboration using Visual Requirements Modeling**

Visual Requirements Modeling has other important characteristics. It addresses the issues that developers might not fully understand the business. The design of accurate and stable requirements cannot be completed until users gain experience with the proposed system<sup>2</sup>. This helps them visualize the actual system they are going to be using, and thus flush out missing requirements, based on their user experience with the application. The business users can define their requirements, without having to understand computer programming. The developers can model these requirements, and the business users can finally validate the requirements by seeing a visual prototype of the application. Enabling a close B2IT collaborative environment, we can facilitate useful negotiations leading to successful software development projects. Modeling is the easiest way of moving through the requirements validation process, to identify missing requirements, validate existing requirements, map requirements to abstract objects and creating usage scenarios. Our process also requires traceability through the requirements, which ensures the support for the evolution of requirements, by identification of missed and new requirements. This allows the business and developers to collaborate to<sup>8</sup> –

- Restate the requirements in a clear, precise and unambiguous manner,
- Identify and correct internal inconsistencies, and
- Test the requirements by proving statements about the expected behavior.

#### **4.2.4. Requirements Modeling Language**

Nobe and Warner describe in their paper, the language used to capture the requirements should facilitate analysis of the model<sup>12</sup>. The requirements gathering language should be standards based and should demarcate clearly between a Requirements Definition phase and the traditional Design Phase. Requirements Definition is not expected to replace a traditional design phase, using a language such as UML.

This defines the need of an integrated Requirements Gathering and Modeling language or tool. We suggest the introduction of some language specification such as a Requirements Modeling Language (RML©). (We know of at least one company that is working on a specification of this sort. However, since the company is in stealth mode, we cannot divulge any further information on this).

## 5. Visual Modeling and traditional Software Development Models

In this section we will look at some of the traditional development processes, and show how the need for rapid visual prototyping and how it can be introduced seamlessly into the SDLC.

### 5.1. Waterfall Model

Figure 1 shows the traditional and proposed Waterfall mythology. From a cost-benefit perspective, even if we assign equal cost to each step now we are iterating over two steps, instead of five steps. Empirically, as published by the Standish Group, and other research firms, the Requirements Gathering Phase is the primary time-hog and number one reason of failure of a software project. Thus, clearly, we need to optimize this phase in terms of time, and at the same time be able to validate the outputs (deliverables) from this phase to be correct.

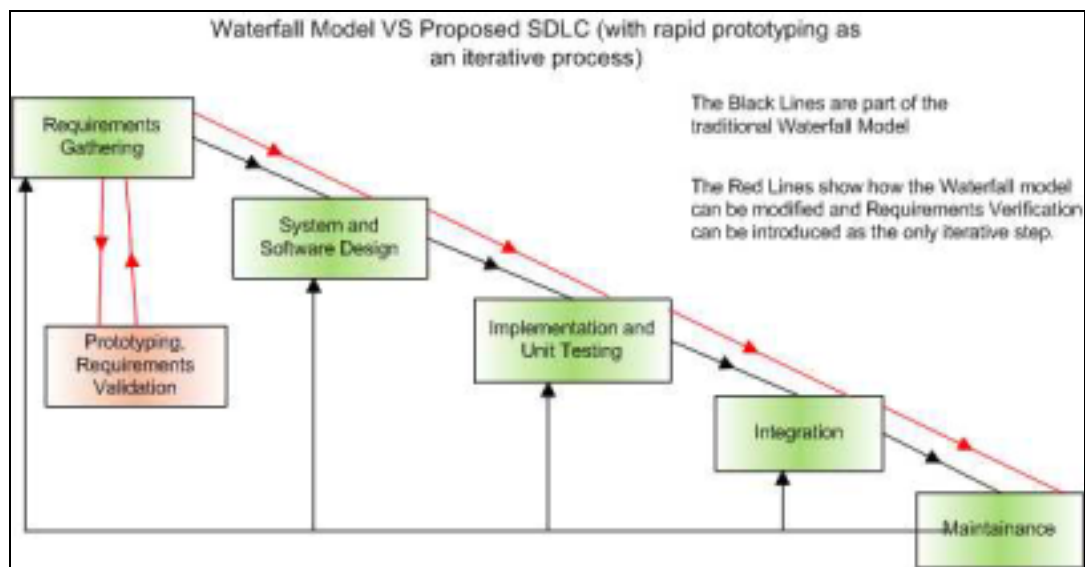


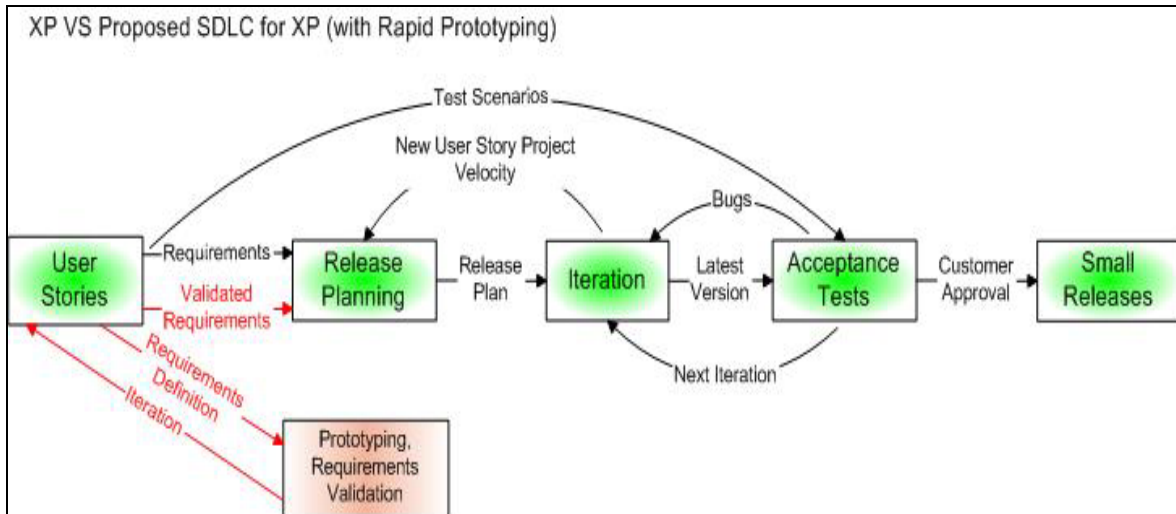
Figure 1: Proposed Waterfall Model

We can see that in the traditional Waterfall model, we iterate over five steps. In the new proposed model, we introduce a step, and then iterate over two steps. Even if we can assign equal cost to each step, with the new model, we are iterating over two steps instead of five, thus clearly we are saving time and money. In practice however, each step cannot be assigned equal weight, and thus the cost savings can be analyzed more quantitatively.

Introducing our Requirements Definition cycles, bring about close B2IT collaboration, which is key to accurately defining the requirements for software, and ensuring the communication of the requirements from the business users to the engineers and developers is clear and correct.

## **5.2. Extreme Programming**

Extreme Programming (XP) is a relatively new approach to software development. About five years old, it has already been proven at cost conscious companies like Bayerische Landesbank, Credit Swiss Life, DaimlerChrysler, First Union National Bank, Ford Motor Company and UBS. XP has been successful because it stresses customer satisfaction. The methodology is designed to deliver the software your customer needs when it is needed. XP empowers developers to respond confidently to changing customer requirements, even late in the life cycle. XP introduces the concept of defining small user stories, using CRC cards, or other definition concepts. This is a good way to capture requirements, and organize them into logical iterative builds, assign priorities and develop these user stories. However, XP does not talk about validation of these requirements. Yes, XP presents an optimized methodology to develop these requirements, but if the quality of the requirements is not addressed, the result might not be satisfactory. Thus, we propose the introduction of our Requirements Definition or Visual Requirements Modeling phase into this process.



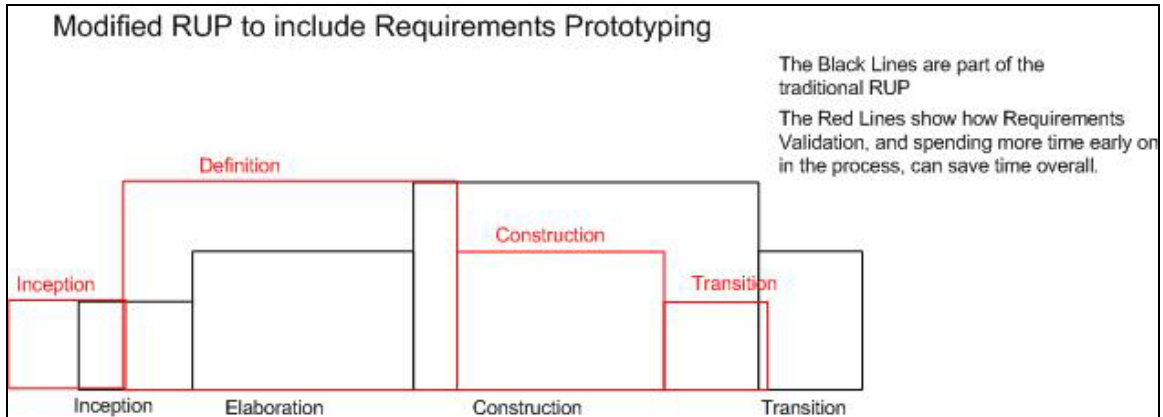
**Figure 2: Modified XP Process**

Figure 2 shows the revised XP model (original XP model taken from <http://www.extremeprogramming.org/map/project.html>), showing the introduction of the iterative Requirements Definition phase, over the User Stories. Once we have short user stories, we can define a more detailed requirements specification by introduce the modeling of these user stories using a prototyping tool. Using comments or other documenting features, we can co-relate the user stories to elements of the model. Refining the user stories as we iterate over the prototype, will enable us to define an accurate, correct, and complete set of user requirements. This ensures an accurate set of validated requirements are passed to the Release Planning cycle.

### 5.3. RUP

Rational Unified Process®, (RUP) is a product offering by Rational Software that provides an out of the box software implementation methodology. The first iterations of RUP are focused on requirements gathering and less on software development. RUP also requires the use the Rational Unified Modeling Language (UML) for system design. RUP also encourages a full prototype during the initial phases of the project that includes all functional and technical layers for the final target system. This can get very complex as

developers attempt to solve all the non-functional requirements during the prototype phase. Although RUP is adaptable to fit the methodology of any organization, it is a very prescriptive, heavy methodology designed around the process of software implementation.



**Figure 3: Modified RUP Process**

The traditional RUP process, as shown in Figure 3, defines the Elaboration phase as the main design phase for software. A majority of the time is the Construction phase, which corresponds to the traditional development phase. Our model proposed modifying RUP, to change the Elaboration Phase to a Definition Phase, and spending more time in this phase. The requirements are complete, accurate, and well understood by the developers. Our claim is that an accurate software definition will reduce the time and cost of the Construction phase, due to having less requirement related problems during User Acceptance Testing (which in this model is a part of the Construction phase).

## 6. Benefits of Visual Requirements Modeling

These two problems, market window and increase complexity puts enormous pressure on IT to deliver applications that meet customer expectations on the first attempt. Time wasted at the front and back end of projects can no longer be tolerated. Operational excellence increases the confidence of both business and IT and contributes to the continual evolution of business solutions.

Requirements are defined by focusing on the end-user experience and the relevant business processes. In developing a solution to the business problem being solved, the end goal is understood in terms of each step the user takes in the process. Eighty percent of use case information can be gathered using informal use cases. The extra effort involved in formalizing use cases gains a small additional amount of information that could easily be gained through direct communication between the developer and the business user. A lightweight, informal requirements gathering approach is used to get as much information as practical to the developers as quickly as possible. As requirements are developed and the communication between business and IT is underway, prototypes (both functional and non-functional) are used as a basis of collaboration between the parties. Business communicates requirements to IT using prototypes, which then become the basis for the development process. Prototyping is very useful for flushing out all major and minor requirements of the application, and gives the business an opportunity to comment and change the end product both before development begins and during the development process. Visual modeling helps this prototype development, by allowing developers and business users to iterate over the requirements and prototype phases, and tie requirements to the prototype. This removes the distinction between the requirements and the prototype phases, and allows more business to IT (B2IT) collaboration. New or changing requirements are easily incorporated into the iterative planning process. In the multi-release approach, new requirements have to wait until the next release cycle, sometimes as much as twenty weeks away. In traditional SDLC, requirements that are

identified after the requirements definition phase are subject to a strict change control process and generally result in higher costs and a delay to the implementation of the system. The iterative Visual Requirements Modeling approach reduces the time for incorporating new or missing requirements from twenty weeks to sometimes as less as one to two weeks.

## 7. References

1. James L. Sidoran, Carla L. Bums Capt Scott Maethner, Dr David Spencer, Hollis Bond, "A Case Study on Rapid Systems Prototyping and its Impact on System Evolution," Proceedings of the Sixth IEEE International Workshop on Rapid System Prototyping (RSP'95), 1995.
2. Luqi, "Software Evolution through Rapid Prototyping" IEEE Software, May 1989.
3. Stephen F, Andriole, "Fast Cheap Requirements: Prototype, or else!" IEEE Software, March 1994.
4. Suzanne Robertson, James Robertson, Mastering the Requirements Process, Addison-Wesley Pub Co; ISBN: 0201360462.
5. Darrell Barker, "Requirements Modeling Technology – A Vision for better, faster, and cheaper systems", Information Directorate (AFRL/IFTA), Air Force Research Laboratory, Wright-Patterson Air Force Base, Ohio.
6. Mei Lu, Xinpei Zhao and Mingshu Li, "Object-Oriented Requirements Modeling Based on UML", Chinese Academy of Sciences
7. Hans-W Gellerson and Martin Gaedke, "Object Oriented Web Application Development," IEEE Internet Computing, Jan-Feb 1999.
8. Steve Easterbrook, Robyn Lutz, Richard Covington, John Kelly, Yoko Ampo and David Hamilton, "Experiences Using Lightweight Formal Methods for Requirements Modeling", IEEE Transactions of Software Engineering, IEEE Vol. 24, No. 1; Jan 1998, pp. 4-14.
9. Luqi, "Knowledge-Based Support for Rapid Software Prototyping", IEEE, Winter 1998.
10. Luigi Lavazza and Guiseppe Valetto, "Enhancing Requirements and Change Management through Process Modeling and Measurement", IEEE 2000.
11. Anthony I. Wasserman, "Requirements for OO Design Environments", Proceedings of the 1995 Software Engineering Environment Conferences (SEE'95).

12. C.R. Nobe and W.E. Warner, “Lessons Learned from a Trial Application of Requirements Modeling Using Statecharts”, Proceedings of the 2<sup>nd</sup> International Conference on Requirements Engineering (ICRE '96).
13. Eric S. K. Yu, “Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering”, Proceedings of the 3<sup>rd</sup> IEEE International Symposium on Requirements Engineering (RE'97).
14. Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite, “Integrating Non-Functional Requirements into Data Modeling”, Proceedings of the IEEE International Symposium on Requirements Engineering.
15. Constance Heitmeyer, James Kirby, Jr., Bruce Labaw, Myla Archer, Ramesh Bharadwaj, “Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications”, IEEE Transactions on Software Engineering, Nov 1998 (Vol. 24, No. 11, pp. 927-948).

## 8. Annotated bibliography

1. James L. Sidoran, Carla L. Bums Capt Scott Maethner, Dr David Spencer, Hollis Bond, "A Case Study on Rapid Systems Prototyping and its Impact on System Evolution," Proceedings of the Sixth IEEE International Workshop on Rapid System Prototyping (RSP'95), 1995.

This paper talks about the requirements gathering phase of the software development life cycle (SDLC) and introduces how prototyping the requirements, early in the SDLC, can have huge costs and time impacts of the evolution of the software. The paper talks about a spiral development model within the requirements engineering process to identify missing requirements and reviews significant improvements to the requirements engineering or specification process.

This paper supports our research on prototyping of requirements to improve testing and validation of requirements to help business users to identify all missing requirements and verify that the captured requirements actually fully encompass the business needs. The paper by Sidoran provides insight to the prototyping process such as a methodology for when and how to apply requirement prototyping. The paper describes a process model within the requirements specification phase (a spiral process model) to increase the chances of achieving a comprehensive set of validated requirements (requirements that meet the needs of the business).

2. Luqi, "Software Evolution through Rapid Prototyping" IEEE Software, May 1989.

This paper describes the evolution of software during the prototyping stages, and presents an iterative prototyping cycle. Luqi claims that often, requirements change even after the initial release of the software. This means, that the software released needs to be modified

or changed, and sometimes even scrapped all together. This paper supports our research on how to introduce a prototyping phase between the requirements and the design phases of the SDLC. It also talks about how a significant number of the requirements changes that generally happen after the initial release of the software, can be caught and addressed during the prototyping stages.

3. Stephen F, Andriole, “Fast Cheap Requirements: Prototype, or else!” IEEE Software, March 1994.

Stephen talks about how the cost of missing or incomplete requirements impacts software. He goes on to talk about how prototyping can reduce this cost significantly. He introduces the concepts of modeling requirements and interactive prototyping. This paper will be developing on his ideas of how effective models can be used to create a visual and interactive prototype of a software application. We expand on the steps within the iterative requirements prototyping cycle as described by Stephen. These steps include elicit, model, prioritize, design and develop requirements.

4. Suzanne Robertson, James Robertson, Mastering the Requirements Process, Addison-Wesley Pub Co; ISBN: 0201360462.

This book is a very in-depth guide on the requirements gathering process, using the Volere Requirements Process Model. The book presents the notion of requirements fitness and validness. It shows how you can validate that software was successful, after it is developed. Our paper will expand these ideas and show how you can determine the fitness and validness of a requirement before the software is developed.

5. Darrell Barker, “Requirements Modeling Technology – A Vision for better, faster, and cheaper systems”, Information Directorate (AFRL/IFTA), Air Force Research Laboratory, Wright-Patterson Air Force Base, Ohio.

This paper talks about the Rosetta System Level Design Language (SLDL). It is described as merely a requirements modeling language. Darrell claims that before Rosetta SLDL came about, there was no good way to model requirements. Requirements related issues were the main source of problems that plagued the software development process. Statistics shows that only 26% of all projects are successful. The same study shows that projects, on average, are over budget by 90% and take 120% of the time as compared to the original estimates. Darrell describes how requirements related issues tend to be the primary cause of these overruns. We agree with his view, and expand on his ideas on how to overcome the problems related with requirements specifications. We emphasize Darrell's ideas on requirements modeling can improve the quality of a system, as a system is described as its conformance to requirements.

6. Mei Lu, Xinpei Zhao and Mingshu Li, "Object-Oriented Requirements Modeling Based on UML", Chinese Academy of Sciences.

The paper by Lu, Zhao and Li, concentrates on how requirements modeling can be done using UML. It introduces the concept of User-driven Domain-specific Requirements Engineering (UDRE) and shows that using the UDRE methodology we can attain a conceptual model of the enterprise as seen by the end-users. They talk about an Object Oriented approach to requirements modeling and the use of the diagrammatic notation of UML.

Our research agrees with some of the concepts described in the paper by Lu, Zhao and Li. They define the requirements process to be iterative and believe in requirements prototyping using a graphical notation, such as UML. We are proposing the graphical modeling and verification of requirements.

7. Hans-W Gellerson and Martin Gaedke, "Object Oriented Web Application Development," IEEE Internet Computing, Jan-Feb 1999.

Gellerson and Gaedke talk about how most web applications are developed without a well-defined development process. The requirements phase is haphazard and thus requirements collected are adhoc. We will expand on their ideas and shows the need to employ a development process to effective create web applications.

8. Steve Easterbrook, Robyn Lutz, Richard Covington, John Kelly, Yoko Ampo and David Hamilton, "Experiences Using Lightweight Formal Methods for Requirements Modeling", IEEE Transactions of Software Engineering, IEEE Vol. 24, No. 1; Jan 1998, pp. 4-14.

Steve's paper realizes the fact that over seventy percent of the software errors found are due to inadequate or missing requirements. He emphasizes that formal design methods of addressing this problem, such as the use of UML, only partially address this problem. We need a formal requirements engineering process to address the issue. Steve shows the example of the requirements process followed at NASA. The consequences of not having a formal requirements engineering process for mission critical operations such as NASA, the health care industry, etc. can be catastrophic. However, most traditional software applications are not that mission critical (we are not dealing with life and death situations, or do not hold responsibilities for the lives of people). Thus, we do not have to be as rigid in our requirements phase as described in this paper. Our paper can draw certain contrasts for the paper by Steve. We can talk about the non-mission critical nature of most software applications, and show trade-offs between hundred percent accuracy in the requirements engineering phase and the level of error that an average software company would be willing to accept.

9. Luqi, “Knowledge-Based Support for Rapid Software Prototyping”, IEEE, Winter 1998.

Luqi introduces the concept of a software prototyping language, and explains how a software prototype can be considered as the initial version of the software. Software Architects use prototypes to verify the quality of the application. They demonstrate the application to customers to get a feel for the missing or misunderstood requirements. The process of defining requirements, adjusting requirements, building the prototype and modifying the prototype based on feedback, can be considered as the iterative process as defined and explained in our paper. Thus, Luqi’s process for requirements prototyping closely resembles the process we are describing for requirements verification and validation.

10. Luigi Lavazza and Guisepe Valetto, “Enhancing Requirements and Change Management through Process Modeling and Measurement”, IEEE 2000.

The paper by Lavazza and Valetto describes a process around the requirements engineering phase of the SDLC. They describe a process that can improve the effectiveness of the requirements stage, provide metrics to quantitatively measure, and analyze the impacts of requirements changes to the development process. Although this paper we do not get into the quantitative benefits of a solid requirements prototyping process, the paper by Lavazza and Valetto can be used a supplement to this paper to describe the process in those terms. We do incorporate the concepts of cost estimation for an improperly managed requirements phase and how that can cause overall budget and time overruns in the delivery of software.

11. Anthony I. Wasserman, “Requirements for OO Design Environments”, Proceedings of the 1995 Software Engineering Environment Conferences (SEE’95).

Anthony attacks the requirements problem from a different angle. He describes the design and requirements for an OO Design Environment that can be used to better the requirements process. He presents some visual techniques such as Intelligent Drawing, and defines a framework for visually designing systems. Our paper also talks about a visual requirements modeling tool, and we expand on Anthony's ideas for navigating a design model, graphical and tabular editing of requirements design environments, and the capability to add formal and informal details to the various design elements.

12. C.R. Nobe and W.E. Warner, "Lessons Learned from a Trial Application of Requirements Modeling Using Statecharts", Proceedings of the 2<sup>nd</sup> International Conference on Requirements Engineering (ICRE '96).

Nobe and Warner claim that engineers that are writing requirements for complex systems generally do not use formal techniques or sophisticated tools to specify correct system behavior. In some cases the requirements documents are huge, like a ten thousand page requirements document for a model aircraft. They define a new process of defining requirements that follows the same basic criteria that we define in our paper. These include the basic assumptions that the requirements engineers need not understand computer programming, the process should not be performed in a laboratory, rather on a white board on the engineers conference room, the language for capturing requirements should be standardized, and the tool used for collection of requirements should provided a dynamic prototyping environment. Our paper addresses and expands on each of these ideas and presents a mythology to do these steps effectively.