

**A Synergistic Approach to Software Requirements Generation:
The Synergistic Requirements Generation Model (SRGM), and
An Interactive Tool for Modeling SRGM (*it*SRGM)**

Rajat Sud

Thesis submitted to the Faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Approved:

James D. Arthur, Chair

Mary Beth Rosson

Stephen H. Edwards

May, 2003
Blacksburg, Virginia

Keywords: Requirements Generation, Software Engineering, Model, Tool

Copyright 2003, Rajat Sud

**A Synergistic Approach to Software Requirements Generation:
The Synergistic Requirements Generation Model (SRGM), and
An Interactive Tool for Modeling SRGM (*itSRGM*)**

Rajat Sud

(ABSTRACT)

The importance of a well-formulated set of software requirements contributing to a successful software development effort has been underscored in recent times. However, the software industry still faces a dearth of process models and artifacts populating the requirements generation process. Of those that do exist, they are often disconnected and narrowly focused, providing little structure to the requirement generation phase. Current methodologies advocate useful guidelines, but do not *enforce* them. To address these concerns, we introduce the “Synergistic Approach to Software Requirements Generation” – an approach composed of two components – a model and an interactive tool implementing the model.

The first component is the *Synergistic Requirements Generation Model* (SRGM). The SRGM is complete with detailed processes spanning the entire software requirements generation phase. These processes have been identified and decomposed to low-level activities with the intent to improve clarity and understanding.

The second component entails the *An Interactive Tool for modeling the SRGM* (*itSRGM*) codifying the structure dictated by the model. The tool *enforces* guidelines and provides visual representations of the progression of activities involved in the requirements generation process.

ACKNOWLEDGEMENTS

I am forever thankful to my advisor, Dr. James D. Arthur for all his guidance and encouragement over the past two years. Working with him has been a gratifying experience. I shall never forget the effort he spent in strengthening my concepts related to the work presented in this thesis *and* life in general.

I am also thankful to the Computer Science department – the faculty and staff. Being here has been an incredible learning experience. I shall always remember the years spent here as a *highpoint* in my life.

I am extremely grateful to my parents, who, without questioning have always provided me the support and encouragement to acquire the education I wanted.

Last but not least, my greetings to all friends and family back home and in the United States, who have positively influenced my life and have shaped the person that I am today.

TABLE OF CONTENTS

1. INTRODUCTION	1
1.1 The Software Industry	1
<i>1.1.1 Chaos</i>	<i>1</i>
<i>1.1.2 Software Development Paradigms.....</i>	<i>2</i>
<i>1.1.3 Shifting focus in the SDLC</i>	<i>4</i>
1.2 The Importance of Requirements Engineering	5
<i>1.2.1 Standish Group</i>	<i>5</i>
<i>1.2.2 The Requirements Generation Process</i>	<i>6</i>
1.3 Current Practices.....	7
<i>1.3.1 Participatory Approaches.....</i>	<i>7</i>
<i>1.3.2 Formal Approaches.....</i>	<i>9</i>
<i>1.3.3 Best Practices</i>	<i>9</i>
1.4 The “Synergistic Approach” to Requirements Generation.....	10
1.5 Problem Statement	11
<i>1.5.1 The Problem.....</i>	<i>12</i>
<i>1.5.2 The Solution (Strategy)</i>	<i>13</i>
1.6 Blueprint of Thesis.....	14
 2. BACKGROUND.....	 16
2.1 Requirements Generation	16
<i>2.1.1 Requirements</i>	<i>16</i>
<i>2.1.2 Requirements Elicitation.....</i>	<i>18</i>
<i>2.1.3 Requirements Analysis.....</i>	<i>19</i>
<i>2.1.4 Requirements Specification.....</i>	<i>20</i>
<i>2.1.5 Requirements Verification and Validation.....</i>	<i>22</i>
<i>2.1.6 Requirements Management</i>	<i>24</i>
2.2 Requirements Generation Models.....	25
<i>2.2.1 A Survey of Existing Requirements Generation Approaches.....</i>	<i>25</i>

2.2.1.1 Methodologies.....	25
2.2.1.2 Models and Frameworks	31
2.2.1.3 Methods	37
2.2.2 A Synergistic Approach? (Holistic support for Requirements Engineering)	44
2.3 Requirements Generation Environments	44
2.4 Development of the Requirements Generation Model	47
2.4.1 Key Issues.....	47
3. THE SYNERGISTIC REQUIREMENTS GENERATION MODEL	51
3.1 Conceptual Overview.....	53
3.1.1 Initial Customer Interaction	55
3.1.2 Document System Characteristics.....	56
3.1.3 Problem Description Document (System Conceptual Overview).....	56
3.2 Problem Synthesis.....	58
3.2.1 Requirements Engineer Education	59
3.2.2 Problem Analysis	59
3.2.2.1 Problem Identification.....	60
3.2.2.2 Problem Decomposition	63
3.2.2.3 Context and Constraints Analysis	64
3.2.3 Needs Generation.....	68
3.2.3.1 Needs Elicitation	69
3.2.3.2 Needs Analysis	70
3.2.3.3 Needs Evaluation	71
3.3 Requirements Capturing.....	72
3.3.1 Customer Education.....	72
3.3.2 Requirements Elicitation Meeting.....	73
The Structured Meeting Approach.....	73
3.3.3 Local Analysis.....	75
3.3.4 Requirements Evaluation.....	76
3.4 Requirements Analysis	78
3.4.1 Global Analysis	78

3.4.2 Requirements Specification.....	81
3.5 Requirements Validation.....	82
3.5.1 Requirements Adherence	83
3.5.1.1 Verify Quality Attributes.....	83
3.5.1.2 Requirements Traceability.....	84
3.5.1.3 Formal Technical Reviews	84
3.5.1.4 Reworking (Editing) Requirements	86
3.5.2 Baseline Requirements (Configuration Control)	86
4. AN INTERACTIVE TOOL MODELING THE SRGM (<i>itSRGM</i>).....	89
4.1 Need for An Integrated Tool	89
4.2 Developing the Interactive Tool for Requirements Generation	91
<i>itSRGM Meta-Requirements</i>	91
<i>itSRGM Design Components</i>	92
<i>itSRGM Implementation</i>	92
4.3 Components of the Requirements Generation Tool	94
4.3.1 Processes.....	94
4.3.2 Visual Process Notations.....	96
4.3.3 Mouse Over Effects.....	97
4.3.4 Detailed Process Descriptions	98
4.3.5 Decision Points and Iterations	100
4.3.6 Preview Pane.....	101
4.4 A Synergistic Approach to Software Requirements Generation – <i>itSRGM</i> in Action (w/ screenshots).....	101
5. SUMMARY AND FUTURE WORK.....	109
5.1 Summary	109
5.2 Contributions.....	111
5.3 Future Work	112

References.....	114
Appendix A: Requirements Generation Model, Artifacts and Templates	118
Appendix B: Requirements Management Tools	145
Appendix C: Requirements Specification Guidelines	154
Vita.....	160

LIST OF TABLES

Table 1.1: Best Practices for Requirements Generation	10
Table 2.1: Requirements Engineering Methodologies	29
Table 2.2: Requirements Engineering Models and Frameworks	36
Table 2.3: Requirements Elicitation Methodologies and Techniques	43
Table 3.1: Sources of Input of the SRGM.....	54
Table 3.2: The Problem Description Document	57
Table 3.3: Potential System Constraints.....	67
Table 3.4: Needs versus Requirements	69
Table B.1: A Qualitative Comparison of Requirements Management Tools.....	147
Table B.2: Organizational Factors.....	148
Table B.3: Checklist for selecting a Requirements Management Tool.....	152

LIST OF FIGURES

Figure 1.1: The Waterfall Model for Software Development	2
Figure 1.2: Prototyping Approach to Software Development.....	3
Figure 2.1: The Requirements Generation Model (RGM).....	31
Figure 2.2: The Requirements Triage.....	32
Figure 2.3: The Knowledge-Level Process Model	34
Figure 2.4: The Win-Win Spiral Model.....	35
Figure 3.1: The Synergistic Requirements Generation Model	53
Figure 3.2: Conceptual Overview	55
Figure 3.3: Problem Synthesis	58
Figure 3.4: Problem Analysis	60
Figure 3.5: Problem Identification.....	61
Figure 3.6: Root Cause Analysis (Fishbone Diagram)	62
Figure 3.7: Problem Decomposition.....	63
Figure 3.8: Context and Constraints Analysis.....	65
Figure 3.9: Needs Generation.....	68
Figure 3.10: Problem Analysis (Methods)	70
Figure 3.11: Requirements Capturing.....	72
Figure 3.12: The Structured Meeting Approach.....	74
Figure 3.13: Local Analysis	76
Figure 3.14: Requirements Quality Attributes.....	77
Figure 3.15: Requirements Analysis	78
Figure 3.16: Global Analysis	79
Figure 3.17: Requirements Validation.....	82
Figure 3.18: Requirements <i>Adherence</i>	83
Figure 4.1: Partial Screenshot of <i>itSRGM</i> , indicating a process description	95
Figure 4.2: Screenshot of <i>itSRGM</i> , indicating the “submit” pane.....	96
Figure 4.3: Screenshot representing “Develop Stakeholder Profiles”	97
Figure 4.4: Screenshot of <i>itSRGM</i> indicating the ‘mouse-over’ effect	98

Figure 4.5: Decision Point for verifying the correctness of the problem	99
Figure 4.6: "Meaningful" Questions	100
Figure 4.7: The start of the Requirements Generation process.....	102
Figure 4.8: Specifying the Type of the Input Source Document.....	103
Figure 4.9: Specifying the location of the Input Source Document	104
Figure 4.10: Screenshot representing “Initial Customer Interaction”	105
Figure 4.11: Screenshot representing “Document System Characteristics”	106
Figure 4.12: Specifying the location of the Problem Description Document.....	107
Figure 4.13: The Completion of Conceptual Overview Process.....	108
Figure A.1: Participant Node Tree	134
Figure A.2: Participant Selection as per the JAD Methodology	134
Figure A.3: Participant Selection as per the PD Methodology	135
Figure A.4: Scheduler Front End	136
Figure A.5: A Schematic Representation of the Email Manager.....	137

Chapter 1 Introduction

1. INTRODUCTION

This thesis presents a Requirements Generation Model, along with an implementation of that model, in the form of an interactive tool, both designed to provide a synergistic approach to the software requirements generation process. This chapter motivates the need for an organized approach to requirements generation, and the reasons for creating a synergistic model and the tool codifying the model.

1.1. The Software Industry

Early experience in building software systems showed that existing methodologies were inept. Major projects were, at times, years late. They cost much more than originally predicted, were unreliable, difficult to maintain and performed inefficiently [Pressman 2001]. The inadvertent delays and significant budget overruns caused software costs to rise, while hardware costs were plummeting. New techniques and methods were needed to control the complexities inherent in large software systems.

After more than 30 years of development, the software industry has made enormous progress in developing reliable software, more often than not, within budgetary and schedule constraints [Sommerville 1996]. We now have a better understanding of various processes involved in software development, and have modeled these processes in the form of software development lifecycles, lending much-needed structure to the overall development process. Nevertheless, many large software projects are still late and over-budget. Moreover, software that is delivered often does not meet the *real* needs of the customer.

In this section, we provide a brief description of the progression of Software Engineering as a discipline and then focus in on Requirements Engineering.

1.1.1 *Chaos*

In his classical epic, The Mythical Man-Month [Brooks 1995], Brooks compared early software development practices to a ‘tar-pit’ in which programmers struggled to complete projects within budgetary and financial constraints, just like the “mortal

struggle of great beasts...the fiercer the struggle, the more entangling the tar...” What emerged as a result of this struggle, were esoteric, hard-to-maintain systems.

Over the years, the enormous improvement in hardware forced a dramatic increase in the complexity of the software. As processing power increased, so did the capabilities expected, hence leading to high(er) expectations from software. These expectations revealed the shortcomings in the approaches that governed software development. No longer could the *hypothetical hacker* stationed in a *remodeled garage*, be entrusted with the development of reliable and maintainable software. It was realized, that like any other discipline, software, too, had to be engineered to ensure reliability and high quality [Royce 1987].

The discipline of ‘Software Engineering’ brought about the identification of phases in software development – an organization that brought structure to the activities performed previously in an *ad-hoc* manner. New tools, artifacts and process models also contributed to the production of reliable software.

1.1.2 Software Development Paradigms

A Software Development Paradigm outlines a *development strategy* that encapsulates processes, methods, artifacts and tools. Software paradigms typically consisted of five activities: Requirements Analysis, Design, Implementation, Integration & Testing and Maintenance [Royce 1987]. Different Software Process Models decompose these ‘generic’ activities in different ways. The timing of the activities varies, as does the outcome.

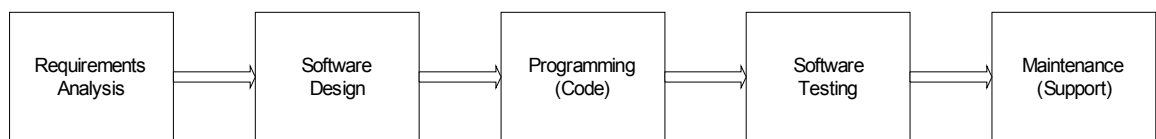


Figure 1.1: The Waterfall Model for Software Development

The first of the development models – The Waterfall Model [Royce 1987] offered means of organizing the development process. This model placed the five activities in a linear timeframe, with each phase following the previous one (Figure 1.1). This

process model places Requirements Analysis process at the start, leading to the development of a *baselined* document – the Software Requirements Specification (SRS). As advocated by the model, the SRS is used to express the system design. The shortcoming of the waterfall model was the inflexible partitioning of the project into distinct stages. It failed to address the *requirements creep* [Carter 2001] problems that were an inevitable consequence of any development project.

The approach that followed was based on the idea of developing an initial implementation (prototype), demonstrating it to the user, and editing, deploying or discarding the implementation based on user response. The intent was to seek user feedback on the prototype and incorporating the feedback in the form of changes/enhancements in the subsequent prototype(s). This approach was referred to as the Evolutionary or the Prototyping Approach to software development.

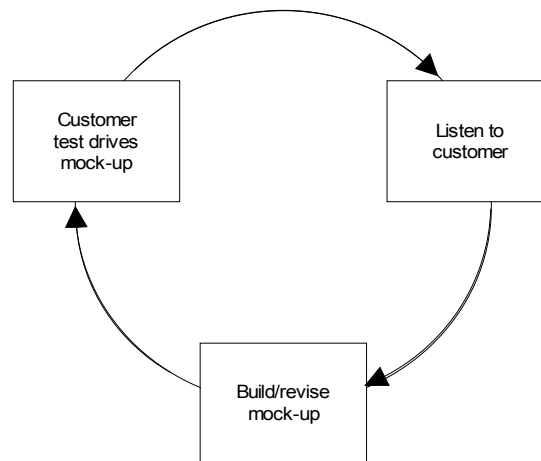


Figure 1.2: Prototyping Approach to Software Development

The Evolutionary Approach, however, suffered from the drawback of consuming time and monetary resources until a correct, compliant product was developed. Reality constraints within the industry rarely afford the luxury to discard fully functional prototypes and re-commence development from the start. Examples of the evolutionary approach to software development include the Spiral Model [Boehm 1988], Rapid Application Development (RAD) [Pressman 2001] and the more recent “Extreme” Programming [Beck 1999].

Although these process models proved to be extremely useful, they focused excessive attention towards the *end product*. The focus was more on developing the *product right*, as opposed to building the *right product*.

1.1.3 *Shifting focus in the SDLC*

A closer examination of existing software development models reveals a common thread: they all contain a Requirements Analysis phase, it is usually one of the first phases in the model, and is composed of a series of activities that relate to the gathering and analysis of requirements from the customer. Although “first” in the sequence of phases, Requirements Analysis has been last in line for re-examination and refinement [Sidky 2002]. In fact, the redefinition and refinement of the original phases defined by Royce proceeded in a backward fashion starting with Integration & Testing. We attribute this to the fact that as we move from Requirements Analysis toward the Integration & Testing phase, conceptualizing and addressing indigenous issues become relatively easier. Testing methodologies¹ and tools were developed providing detailed coverage and automating the testing process, Integrated Development Environments² (IDEs) were conceived to aid programming and debugging activities. Design notations, such as the Unified Modeling Language (UML), and tools were developed standardizing system design representations. In the light of all these CASE tools, support for Requirements Engineering was left out. Moreover, when the software engineering community did begin examining and refining the Requirement Analysis phase, its somewhat inappropriate name contributed to the perception that activities therein, like Problem Analysis and Requirements Elicitation, were only minor ones [Davis 1993]. Subsequently, only within the last few years have we seen a meaningful refinement of “Requirements Analysis,” recognizing the major activities underlying requirements generation.

¹ An elaborate suite of testing methodologies is provided in [Pressman 2001]

² IDEs provide an integrated set of tools and artifacts supporting a process; e.g. Microsoft’s Visual Studio

1.2 The Importance of Requirements Engineering

“The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems.

No part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later”

- F.P. Brooks, [Brooks 1987]

The importance of a well-defined, effective set of requirements engineering activities is constantly being reinforced as we begin to recognize the manifold relationships between the quality of a product and the quality of requirements from which it is developed. In their early empirical study, Bell and Thayer [Bell 1976] observed that inadequate, inconsistent, incomplete, or ambiguous requirements have a critical impact on the quality of the resulting software. Noting this for different kinds of projects, they concluded that “the requirements for a system do not arise naturally; instead, they need to be engineered.” Boehm [Boehm 1988] estimated that the late correction of requirements errors could cost up to 200 times as much as correction during requirements engineering. Therefore, the most important function that the software builder performs for the client is the iterative extraction and refinement of the product requirements.

In this section, we describe the Standish Group findings to emphasize the importance of requirements engineering. The main activities supporting the requirements generation process are also explained briefly.

1.2.1 Standish Group

In the CHAOS Report [Standish 1995], the Standish Group revealed that five of the top eight reasons why projects fail, are related to requirements – incomplete and/or erroneous requirements, lack of user involvement, unrealistic customer

expectations, and requirements volatility. In another survey, it was revealed that the main factors attributing to project *success* were, amongst other factors, user involvement and a clear set of requirements. Software projects that were ‘challenged’ for completion suffered from lack of user input, incomplete requirements specifications, requirements volatility and unclear objectives. These factors hint towards the importance of a complete and clear set of requirements for a project to be successful. In other words, the following criteria need to be an intrinsic part of any successful software development effort:

- A ***Participatory Approach*** to developing software, engaging the user actively throughout the process, especially while understanding and eliciting the needs and requirements
- A ***clear and complete statement*** of requirements, providing an unambiguous ‘common ground’ for developers and users
- ***Proper planning*** and realistic objectives set within smaller timeframes and frequent project milestones
- Constant ***Verification*** and approval from the user about the ongoing progress
- ***Management support*** and a competent, hardworking staff

To reiterate the importance of requirements, the three of the five factors directly relate to having an unambiguous requirements specification. Thus it is imperative that any requirements generation approach intrinsically incorporates these ‘adages.’

1.2.2 The Requirements Generation Process

The Requirements Generation process is placed at the beginning in most software development lifecycles. In itself, the process is composed of the activities listed below.

- ***Requirements Elicitation*** – software requirements are elicited from people or derived from system requirements. An important precursor to the elicitation process is the Problem Synthesis process, during which the underlying issues are *diagnosed* and customer needs are elicited.
- ***Requirements Analysis*** – (adapted from [Brackett 1990]) is usually performed before the customer commits to the actual development process (design and

implementation). The customer assesses the acceptable level of risk regarding the completeness, correctness, and technical and cost feasibility.

- *Requirements Specification* – requirements elicited and analyzed in the preceding phases are documented and expressed in the form of a formal document (deliverable), often referred to as the Software Requirements Specification (SRS).
- *Requirements Verification and Validation* – ensures that the requirements elicited and specified in the SRS adhere to the customer needs or the high level requirements. The requirements elicited are presented to diverse audiences for review and approval. Adherence of specified requirements to pre-defined quality attributes is also tested.
- *Requirements Management* – Any process must incorporate a management methodology that pervades through the entire process and facilitates communication effectively, in case any changes are made. In the words of Leffingwell and Widrig [Leffingwell 2000], Requirements Management is “a process that establishes and maintains agreement between the customer and the project team on the changing requirements of the system.”

These processes are discussed in greater detail in Chapter 2, Section 2.1.

1.3 Current Practices

As revealed in the previous section, deficient requirements are the single biggest cause of software project failure. Despite knowing the importance of a sound requirements generation methodology and its bearing on the overall development effort, there is little that has been done to provide *holistic* support to the whole process.

In this section, we provide a brief description of existing practices of requirements generation activities. Two popular, but distinct, approaches are discussed. The section concludes with a table, adapted from [Hofmann 2001], indicating a summary of key features desirable in a requirements generation methodology.

1.3.1 Participatory Approaches

Current methodologies employ participatory approaches primarily for the requirements elicitation process. Joint Application Design (JAD) and Participatory

Design (PD) are well-known methodologies for operationalizing user participation. Both JAD and PD focus on facilitated interactions between users and developers, employing dynamic group techniques for eliciting and refining requirements. They differ in points of session participant selection, role of facilitators, team orientation and development speed.

PD encourages “intrinsic” interaction between the customer and the requirements engineer throughout the development lifecycle until the final product deployment. The goal of PD is to narrow the “knowledge gap” between the customer and the requirements engineer. PD has a significant advantage of instilling in the customer, ‘a sense of ownership’ of the final product, by actively seeking customer input in all phases of development. PD also requires the requirements engineer to work closely with the customer, learning the customer’s job and the environment in which the proposed system is to be deployed. However, PD suffers from the disadvantage of posing a high initial cost due to the time added to development as both parties become acquainted with each other’s tasks [Carmel 1993].

JAD is based on facilitated requirements-gathering sessions involving the customers and the developers. It is aimed at extracting the greatest amount of pertinent information from all concerned parties. However, a good facilitator is necessary to make JAD sessions successful. A disadvantage to JAD is that, after all the information has been gathered by the facilitator, the right set of people still must meet to sort through the acquired information to establish the correct set of requirements [Groener 2002]. JAD meetings may also be impractical, since workshops may tie up stakeholders for several days at a time. Further, dynamic groups management techniques are required to ensure that the meetings serve their objectives. The facilitator must be experienced; else the meeting may not yield the desired outcome.

JAD and PD are both good approaches to requirements elicitation. However, they do not provide holistic support to the process of requirements generation. An in-depth incorporation of JAD and PD in the Synergistic Software Requirements Generation Model has been discussed in Chapter 3.

1.3.2 Formal Approaches

Requirements Generation methodologies are mostly thought out to be ‘customer-oriented,’ attempting to capture the customer’s intent; the final output, however, provides a basis for implementation, and hence must be represented clearly and unambiguously. Advocates of formal approaches argue that it is imperative that correctness be *proven* before transitioning into implementation. Formal approaches to requirements generation are based on the use of mathematical techniques to prove correctness.

Requirements Definition Languages (RDLs), such as SREM [Alford 1977] aim to overcome the shortcomings of natural language approaches. Shortcomings, such as ambiguity, incompleteness, readability, organization and understandability are minimized through formalization. Proponents claim that formalization can automate verification and validation processes, and can also automate design and coding to some extent.

The advantages of Formal approaches cannot be refuted. When applied correctly, they can almost always assure the correct system. However, more than representation, the concern is *what good a correct system is, if it does not do what the customer needed it to do?* While system correctness is certainly important, there are some disadvantages to formal approaches [Groener 2002].

- Most requirements engineers and practitioners are unfamiliar with formal techniques, nor can they apply them
- Formal Specifications are esoteric to understand, especially for customers
- Formal Specifications ensure that the software meets the requirements; they do not ensure the requirements meets the customer needs

1.3.3 Best Practices

The following table, adapted from [Hofmann 2001], provides a brief insight into the best practices recommended for any requirements generation model. These features were crucial in the development of the Synergistic Requirements Generation Model.

<i>No.</i>	<i>Best practice</i>	<i>Key Benefit</i>
1.	Involve Customers and Users throughout the Requirements Generation Process	Better understanding of the “real” needs
2.	Identify and consult all likely sources of requirements	Improved requirements coverage
3.	Allocate 15 to 30 percent of total project effort to Requirements Generation activities	Maintain high-quality specification throughout the project
4.	Provide specification templates and examples	Standardization of deliverables and improved quality of specification
5.	Prioritize requirements	Focus attention on the most important customer needs
6.	Maintain traceability across all process deliverables	Explicit link between requirements generated at the end of process and intermediary artifacts and deliverables
7.	Use peer reviews, scenarios, and walk-throughs to validate and verify requirements	More accurate specification and higher customer ownership and satisfaction

Table 1.1: Best Practices for Requirements Generation

1.4 The “Synergistic Approach” to Requirements Generation

The Software Requirements Specification is more than simply just the deliverable obtained from the first phase of a software development lifecycle. It not only helps designers and programmers envision the product, but also dictate all the subsequent activities, including testing, validation and acceptance. Most importantly, it reflects the customer’s intent!

As mentioned in Section 1.1.3, a variety of tools exist for supporting the software development effort. Integrated Development Environments provide editing, developing, compiling and debugging capabilities for programming, design notations exist for expressing system design, and automated tools support testing activities. However, even

though the importance of requirements engineering is acknowledged [Standish 1995], very few tools and *holistic* process models offer support and guidance to the activities defined within this phase. Moreover, current approaches provide *disconnected* support for requirements generation. The existing tools offer minimal support for requirements engineering *only after* the requirements are determined (requirements analysis) and not for actually generating the requirements.

The “Synergistic Requirements Generation Model and Tool” offers a synergistic approach to the requirements generation, which leads the requirements engineer through the process culminating in the production of the Software Requirements Specification. This synergistic approach to requirements generation not only provides tools and artifacts to capture requirements, but also embodies a model, complete with guidelines and protocols. If adhered to, the model ensures the production of an unambiguous and complete requirements specification, which most importantly will also capture the customer intent.

The model can be viewed as a framework providing a definitive set of activities, and a guiding structure. This framework is codified in the form of an “interactive tool” potentially providing the requirements engineer with a set of executable tools and artifacts and a visual context within which requirements generation occurs. The tool starts with activities related to the identification of the *real* problem underlying the need for a change, followed by elicitation and analysis activities. The final output produced by the tool is a *validated* set of software requirements specification. In short, the tool provides a guiding structure to the requirements generation activities, previously conducted in an *ad hoc* manner.

The model is described in Chapter 3, while the interactive tool, along with screenshots, is presented in Chapter 4 of this document.

1.5 Problem Statement

To address the issues discussed and to create a *holistic* approach to the requirements generation process, we have developed a model and an interactive platform, consisting of tools, artifacts, visual representations and guidelines. This platform embodies a model, a

detailed description of which is provided in Chapter 3. This section organizes the thesis in the form of a Problem-Solution scenario. We start by discussing the problem at hand, which essentially is a summary of the information related to the shortcomings of current approaches and the motivation for a holistic approach. The solution provides an overview of our model and tool and the manner in which it addresses the problem.

1.5.1 *The Problem*

The contributions and approaches discussed in the previous sections all aim to deliver an accurate set of requirements specification. As is evident, they offer (disjointed) support to the requirements generation process. Specifically, the following key issues must be addressed:

- Non-synergism – Current approaches provide *disconnected* support for requirements generation activities. A model should incorporate the “best of” approaches into one singular process
- Holistic Model – A model should account for all phases of the requirements generation lifecycle, *i.e.* it should have sub-processes and/or activities defined for requirements elicitation, analysis, specification, verification and validation, and management
- Complete *in-depth* categorization of processes – Current processes do not incorporate detailed guidelines and activities to be performed. A model must describe, in detail, all processes and activities
- Guidance – Merely specifying the activities is *futile*, unless the practitioner is actually led through the process, via means of *enforced* structure and *explicit* guidelines

For example, JAD and PD offer guidance for eliciting information from stakeholders. Existing tools such as ARM (Automated Requirements Management) [NASA 1997] provide support for requirements specification, acting as a text parser identifying ambiguous terms in the SRS. Other tools³, such as those mentioned in [Weigers 1999] provide elaborate traceability features (e.g. Rational® RequisitePro®)

³ A detailed survey of 6 popular requirements management tools is provided in Appendix B

that aid in tracking changes. Many tools also provide analysis methodologies to perform “*what if*” analysis (e.g. Caliber RM).

All these tools lack the presence of an overarching structure dictating the flow of activities to be performed, and the context in which each of the tools can be employed. Furthermore, because of the narrow focus of these tools, even when a quality set of requirements is generated, the customer’s intent is often not met. Therefore, the ultimate goal of any requirements generation methodology or tool should be to capture and specify high-quality requirements adhering to the customer’s intent.

1.5.2 The Solution (Strategy)

Our vision is to create a requirements generation ‘environment’ that provides tools and artifacts to capture requirements, and also embodies a model, complete with guidelines and protocols. If adhered to, the model ensures the production of an unambiguous and complete requirements specification, which, most importantly, will also capture the customer intent.

The Model

The model will comprise a framework providing a definitive set of activities. The model will be expressed at multiple abstraction levels providing information (guidelines) about low-level activities, and will lead the requirements engineer through the requirements generation process. The model will span the entire requirements generation process, addressing the activities of requirements elicitation, analysis, specification, validation and requirements management.

We have also identified an additional phase – Problem Synthesis, within requirements engineering. This phase deals with identification and analysis of the problem that the proposed system must address. The activities enumerated within problem synthesis will precede the elicitation of requirements.

The model incorporates the following features:

- Decomposition of the requirements generation process to provide varying levels of detail
- Elaborate set of guidelines aiding the process progression

- Collaborative, Participatory approach involving diverse, but pertinent, stakeholders
- Constant verification and validation activities ensuring that the customer's intent is always captured and verified.

The Synergistic Requirements Generation Model is conceived as per the Waterfall Model, during which requirements are generated upfront, at the start of the development lifecycle. We have characterized requirements such that it is performed before design. The SRGM, in its current form, is not applicable to other approaches, such as the Spiral Approach [Boehm 1988] and the Evolutionary paradigm [Pressman 2001]. However, it may be modified into lightweight versions, to be applied to the incremental approach to software development.

The Interactive Tool

The tool 'codifies' the activities described in the model. It leads the user (typically the requirements engineer) through the requirements generation process by providing appropriate guidelines, tools and artifacts at every step. It starts with activities related to the identification of the *real* problem motivating the need for a change, followed by elicitation and analysis of requirements. The final output produced by the interactive tool will be a *validated* SRS. In short, the tool will provide a guiding, *synergistic* structure for requirements generation activities that were previously conducted in an *ad hoc* manner.

1.6 Blueprint of Thesis

The remaining chapters of this thesis are set as follows. Chapter 2 discusses the background of this thesis. We provide an introduction to 'Requirements Engineering,' and identify existing processes within it. We focus on existing Requirements Generation Models, leading to the identification of the key motivating factors for developing the Synergistic Model for Software Requirements Generation. We also provide a brief survey of existing environments, recognizing the need for an integrated, synergistic tool that will be easy to use and can be incorporated in any development effort. Chapter 3 presents the details of the Synergistic Model for Software Requirements Generation, highlighting the

holistic point of view. Chapter 4 focuses on the codification of the Model, described in Chapter 3, in the form of an interactive tool. A final summary, concluding remarks, and suggestions for future work close the thesis in Chapter 5. Following the last chapter are references, appendixes and the author's vita.

Chapter 2 Background

2. BACKGROUND

This chapter presents the background information that has had a major influence on the research presented in this thesis. The information presented focuses on developments, methodologies and methods in the field of Requirements Engineering.

Section 2.1 provides information about the phases involved in the requirements generation process. It is imperative that all these phases be included and *accounted* for in a *holistic* requirements generation model. Section 2.2 presents a survey of existing requirements generation methodologies, models, frameworks and methods. While developing the ‘Synergistic Model for Requirements Generation (SRGM),’ an attempt was made to extract salient features from these models and incorporate them into the SRGM. Finally, Section 2.3 outlines key issues to be addressed for a requirements generation model, which *lays the foundation* for the SRGM.

2.1 Requirements Generation

This section provides pertinent background information about the components of Requirements Engineering (RE). We start by providing the IEEE definition of requirements, adapted from IEEE Standard Glossary of Software Engineering Terminology (ANSI/IEEE Std. 729-1983) and then explain the components of the RE process.

2.1.1 Requirements

Software Requirements can be defined in several different ways:

- A software capability needed by a user to solve a problem or achieve an objective
- A software capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document
- A short description sometimes used in place of the term “software requirements specification.”

Jones suggests that requirements are “the statement of needs by a user that triggers the development of a program or system” [Jones 1994]. Alan Davis [Davis 1993] broadens this concept to “a user need or a necessary feature function, or attribute of a system that can be sensed from a position external to the system.” These definitions emphasize *what* will go into the product, distinct from *how* the product will be designed and constructed. The following definitions moves further along the spectrum from user needs to system characteristics [Sommerville 1997].

Requirements are...a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.

These diverse definitions indicate that no clear, unambiguous understanding of the term “requirements” exists. The real requirements actually reside in the people’s minds [Weigers 2001]. Any documented form of the requirements (such as an SRS) is just a model, or representation [Lawrence 1996] reflecting peoples’ needs. We need to ensure that all project stakeholders arrive at a shared understanding of the terms used to describe these requirements.

The inherent difficulty in obtaining a precise, unambiguous definition of requirements relates to viewing ‘requirements generation’ as a process. First attempts to structure the requirements gathering process can be traced back to the time when it was simply referred to as “Requirements Analysis” [Pressman 2001]. The term “Requirements Analysis” almost always overshadowed the important activities involved before analysis was actually conducted. This process assumed that requirements had already been elicited and suggested sub-processes aiding *only* analysis. Although a misnomer, the division of activities and sub-processes was a novel attempt to structure the overall process.

Requirements Analysis was originally divided into five areas of effort: (1) problem recognition, (2) evaluation and synthesis, (3) modeling, (4) specification, and (5) review [Davis 1993]. These processes paved the way into dividing the “Requirements Engineering” process into definite activities and sub-processes. They are (1) requirements elicitation, (2) requirements analysis, (3) requirements specification (4) requirements verification and validation, and (5) requirements management. We discuss each individually in the sections that follows.

2.1.2 *Requirements Elicitation*

Requirements Elicitation can be thought of as a process through which the customers (buyers and/or users) and the developer (contractor) of a software system discover, review, articulate, and understand the users' needs and the constraints on the software and development activity [Lauesen 2002]. In essence, elicitation is the process of finding and formulating requirements.

Requirements Elicitation involves the gathering of information related to the overall goals of the system, the present work and problems, and detailed issues that the system must address. Various elicitation approaches are discussed in Section 2.2. Any selection should be made based on the project type, schedule and budgetary constraints. For example, the approach for requirements elicitation for a shrink-wrapped product will differ from an effort to build a custom system. In case of the former, a set of potential software users, not bearing any relation to each other, will be selected to aid the elicitation process. Questionnaires, Interviews, Focus Groups, etc. may be convened to obtain a set of desirable attributes to ensure that the product will be a success. On the other hand, for a customized system, the set of stakeholders will comprise the employees of the organization where the system will be deployed. Requirements will be elicited from these stakeholders. The context of deployment and the work practices of users, specific to the proposed system, will have to be studied.

As an activity, the elicitation of requirements poses myriad problems. Lauesen succinctly outlines these problems in [Lauesen 2002]. Stakeholders have difficulty in expressing what they need! Sometimes they feel they have a problem, sometimes they don't. Even if they see the problem, it may not be easy to express it as a need or a requirement. During task analysis, many users have great difficulty explaining what tasks they perform, and even more difficulty in explaining why they carry out these tasks. A common problem while interacting with stakeholders is that they specify a solution instead of a need. Often the solution may misguide the requirements engineer in actually determining what the stakeholders actually *need*, as opposed to what they *want*. Conversely, once the elicitation process commences, the requirements engineer can obtain a vast number of requirements. Some of them may be essential while others may

be categorized as “nice to have.” It can be difficult to have all stakeholders agree on what is essential and what is a luxury. There are occasions when stakeholders find it difficult to adjust to change. Imagining the consequences of doing a familiar task in a proposed new way can be a daunting task. This attitude may have dire consequences on the overall project, with stakeholders and users possibly rejecting proposals due to general resistance to change.

The difficulties outlined above are inevitable in any development effort that involves requirements elicitation. While no process model or elicitation technique can claim to completely rid problems related to elicitation, they may be minimized by instigating a *participatory approach*, in which the requirements engineer, the customers and the users work synergistically to elicit requirements. All stakeholders must be involved and updated on requirements, as new ones are elicited and existing ones change. This will ensure greater agreement on the requirements, and finally the entire product.

2.1.3 Requirements Analysis

Requirements Analysis is the process of analyzing customers’ and users’ needs to arrive at a definition of software requirements. Requirements Analysis ensures that a *correct* software product is being built. Based on requirements generated from the elicitation process, activities are performed that evaluate the ‘fate’ of the software product, from various standpoints. A development effort not incorporating this phase will be subject to considerable risk, perhaps even resulting in the loss of customer interest; one of the most commonly cited reasons for failed projects [Standish 1995].

As indicated before, when the requirements generation component of the Software Development Lifecycle was first conceived, the term “Requirements Analysis” was often applied to the entire requirements generation process. When the software engineering community began refining the then Requirements Analysis phase, its somewhat inappropriate name contributed to the perception that activities therein, like Problem Analysis and Requirements Elicitation were only minor ones [Davis 1993]. Subsequently, only within the last few years have we seen a meaningful refinement of “Requirements Analysis” that recognizes the major activities that underlie requirements generation. Thus

the term “Requirements Analysis” more aptly represents one of the *constituents* of the Requirements Generation process, as opposed to the entire process itself.

Activities defined within the requirements analysis phase include risk analysis, cost & schedule analysis, market analysis and feasibility analysis. Requirements Analysis also associates attributes, such as status, priority, effort and risk, for each requirement. It is especially important to understand the risks associated with implementing each requirement, including conflicts with other requirements, dependencies on external factors, and technical obstacles. Attributes associated with risk, priority and effort provide additional information about requirements when global analysis activities are conducted, ensuring that a more informed decision can be taken about the project.

Multiple approaches exist for analyzing requirements. It is often helpful to represent a portion of the requirements in more than one way, such as in both textual and graphical forms. Analyzing these different views will reveal insights and problems that no single view can provide [Davis 1993]. Analysis also involves interacting with the customers and users to clarify points of confusion and to understand which requirements are more important than others. The goal is to ensure that all stakeholders arrive at a common understanding of what they will have when the software is deployed [Weigers 2001].

2.1.4 Requirements Specification

Requirements Specification involves the development of a document that clearly and precisely records each of the requirements of the software system. This phase results in the culmination of the “Software Requirements Specification” (SRS). The SRS represents the customers’ requirements, serves as a contract between the development team and the customer, and, finally, it provides information to the developers about designing and implementing the system.

A good SRS should provide a beneficial platform for:

- Establishing the basis for agreement between the customers and the suppliers on what the software product is to do
- Reducing the development effort (careful capturing and documentation of requirements reduces later redesign, recoding and retesting)

- Providing a basis for estimating cost and schedules
- Providing a baseline for verification and validation.

Weigers [Weigers 2001] advocates that all requirements contained in the SRS must be traceable to its origin to ensure that all stakeholders know why every functional requirement belongs in the SRS. The origin may be a scenario, use case, idea, a business rule, a government regulation or some other external source. A requirements traceability matrix can augment the process of tracing every requirement to its source. Further, to enhance the overall organization, every requirement in the SRS must be labeled. “The convention must be robust enough to withstand additions, deletions, and changes in requirements over time” [Weigers 2001].

The requirements stated in the SRS can be presented in a variety of formats and styles. Prototyping is an approach used often for the purposes of analysis and verification of elicited requirements. In general, any application that creates visual displays or interacts heavily with a user is a candidate for prototyping [Pressman 2001]. However, the requirements engineer must ascertain the complexity involved in developing the prototype with respect to the potential gain. In case that the prototype is too complex, another form of specifications may have to be adopted, such as a context diagram. Context Diagrams, as indicated by Lauesen [Lauesen 2002], “shows the product as a black box surrounded by user groups and external systems with which it communicates.” The kind of notation is similar to a dataflow, with the difference being that context diagrams list real world entities, while data flow diagrams represent the flow of data in the design. The context diagram gives an excellent overview of the required product interfaces. It reflects an early decision about the scope of the product. They are almost, always, supplemented with textual descriptions, collectively forming the specification. Context diagrams possess the advantage of aiding verification and validation. They provide an overview of the interfaces and serve as a checklist over what to develop. Also, most customers readily understand the context diagram, and help in spotting incorrect or missing interfaces and/or system components.

The most common approach to representing requirements is listing them as feature specifications, i.e. in textual form. They are usually straightforward to specify and follow the use of *key expressions*, such as, “*The system shall...*” Feature Requirements are appropriate if the customer knows the user tasks very well. If specified correctly, they present an easy-to-understand specification, which can be referred to by customers and developers alike. However, if the document is not structured appropriately, validation may prove difficult since searching features and checking them may be cumbersome for large specifications.

Use cases, scenarios and task descriptions provide an alternative to feature specifications and prototypes. They entail a “story-like” description of the proposed system in use [Lauesen 2002]. A description of users actually using the system provides visible benefits to all stakeholders and eases validation. Developers can easily understand the task descriptions and have an insight into the implementation and the functioning of the system. On the other hand, translation to design may prove to be difficult since the descriptions lack a formal structure. Moreover, extracting relevant information from scenarios is time-consuming.

The approaches discussed above are all viable means to represent the software requirements. An ideal SRS should contain a blend of approaches; the requirements engineer deciding which aspect of the system should be represented with which method.

2.1.5 Requirements Verification and Validation

Requirements Verification and Validation (V & V) is a two-part process. The first part ensures that the software is ready to proceed into design with a high degree of confidence; while the second part ensures that the customers’ intent is accurately captured. V & V activities ensure that the requirement statements are accurate, complete, and demonstrate the desired quality characteristics [Weigers 2001]. A formal inspection of the requirements document is conducted during verification and validation. Verification entails examination as and when requirements are generated, while Validation is conducted on the entire specification document. Informal

preliminary reviews during requirements elicitation are also useful. Weigers [Weigers 2001] provides activities commonly conducted during V & V:

- Ensure that the SRS correctly describes the intended system behaviors and characteristics
- Ensure that the software requirements are correctly derived from the system requirements, or other origins
- Ensure that the requirements are complete and of high quality
- Ensure that all views of the requirements are consistent, and finally
- Ensure that the requirements provide an adequate basis to proceed with product design, construction and testing.

Technical reviews provide a novel approach to conducting requirements verification and validation. Informal review approaches [Weigers 2001] involve the distribution of the work product to peers for the purpose of assessment. In this case, the assessor is expected to describe the product and solicit comments. Informal reviews are good for educating other people about the product and getting unstructured feedback. Whereas informal reviews are performed in an *ad hoc* fashion, a formal review follows a well-defined process with a prescribed sequence of steps. A formal review results in a written report that identifies the material reviewed, the reviewers, the review team's judgment as to whether the product is complete or needs further work, and a summary of defects and issues found.

Requirements V & V offers the biggest potential saving to software development effort, as compared to V & V conducted in design, coding or testing phases. It can uncover and correct many deficiencies that otherwise can go undetected until late in the development cycle, where correction would be much more expensive. Careful V & V prevents problems from rippling to the other phases.

2.1.6 Requirements Management

Requirements Management involves establishing and maintaining an agreement with the customer on the requirements for the software project. The agreement forms the basis for estimating, planning, performing, and tracking the project's activities throughout the software life cycle.

The key to requirements management is *communication*. A good requirements management process helps ensure a high level of communication between stakeholders. For the developers to fully understand the needs of the customers, they must have an open channel of communication with the customers.

Communication is also crucial when requirements change, as they do in any project. Once changes have been agreed upon, they must be incorporated into the project scope, and be communicated to developers and customers, as well. An effective means of communication is thus essential to getting the project right the first time and avoiding expensive rework later in the development cycle.

Requirements Management is crucial to manage inevitable changes brought about in the specification, due to revelations in the other phases of the software development lifecycle. Effective change management requires a process for proposing changes and evaluating the potential cost and impact of the change on the project [Weigers 2001]. In addition, the status of every requirement should be tracked as development progresses through implementation and testing.

Configuration Control is another important facet of requirements management. Frequent baselines should be established during the requirements generation process. As mentioned before, an elaborate change control process must be established and adhered to, ensuring that the change is effected smoothly, without any repercussions to other aspects of the specification. Lastly, it would be useful to employ a requirements management tool, the services of which pervade through the entire software development process. A description of six popular requirements management tools is provided in Appendix B.

2.2 Requirements Generation Models

This section provides background information pertinent to the work presented in this thesis. Each sub-section is concluded with a summary presenting the highlights of each approach and a mapping to the phases on the RE process (Section 2.1). We provide a survey of existing requirements generation approaches, describing existing methodologies, models, frameworks and techniques.

2.2.1 *A Survey of Existing Requirements Generation Approaches*

A number of unique approaches exist for software requirements generation. A survey of a few of these is enunciated in this section. Each approach possesses some unique characteristic or quality that makes it appropriate for the specification of some aspect of requirements. Unfortunately, current tools for requirements engineering usually support only one or two approaches. The ultimate goal of this research is to develop a requirements generation environment that is independent of any particular requirements approach, notation or method. The main objective is to provide a guiding framework that can incorporate any methodology, as dictated by the circumstance or choice.

2.2.1.1 *Methodologies*

This section surveys a selection of existing methodologies for requirements engineering, adapted from [Hofmann 1993]. A categorization and comparison of these approaches is provided at the end of the section.

2.2.1.1.1 *CORE – Controlled Requirements Expression*

CORE is a “prescriptive, top-down” method [Systems 1985]. It identifies clients and their viewpoints, and describes dynamic aspects of the software system with the primary aim to determine functional requirements. CORE assumes that different viewpoints are necessary to specify requirements. Viewpoints are the set of interests to be supported or influenced by the proposed system. Example viewpoints are environment view (e.g. user interfaces), process view (e.g. project schedules, information flows), and reliability view (e.g. failure conditions). CORE defines responsibilities for clients and requirements analysts and structures their

communication. The customer is responsible to resolve conflicting views. The requirements analyst is responsible for the documentation of requirements specifications, which express (multiple) viewpoints, and ensure their successful combination into a coherent specification of the software system.

CORE proceeds iteratively, first defining the problem, then decomposing it into viewpoints, and acquiring information about each viewpoint. This recurrent process leads to a hierarchy of viewpoints. The RE process comprises viewpoint structuring, tabular collection, data structuring, single and combined viewpoint modeling, and constraint analysis.

2.2.1.1.2 IBIS – Issue-based Information Systems

IBIS assumes that RE is fundamentally a conversation among clients and requirements analysts in which they pool their respective knowledge and viewpoints to specify the software system. It organizes and tracks the rationale underlying requirements. The method helps to point out where alternatives have not been considered and where specifications are strong and weak, based on supporting or objecting arguments.

IBIS acquires requirements by keeping track of issues being discussed, propositions on these issues, and arguments in support of a position or object to it [Conklin 1988]. A typical IBIS method scenario would be a stakeholder posting a issue containing a question such as “How should we do X?” Another stakeholder may respond by proposing one way to accomplish X and can also post arguments supporting the proposal. Other stakeholders can post competing arguments or support the proposal. Hence the goal of such discussion is to understand each other’s proposals and to persuade others of one’s viewpoint.

2.2.1.1.3 Jackson System Development

Jackson System Development (JSD) method comprises software development starting from problem analysis and ending with code. The method does not subscribe to traditional life cycle models. Instead of top-down, it develops functional specification middle-out from an initial set of requirements. JSD follows five steps to specify

requirements: entity-action & entity structuring, initial modeling, function step, information function step and system timing [Cameron 1986].

The first two steps are concerned with creating a model of the application domain, the others with the specification of functions to be added to the model. The model of the application domain is created in terms of actions and entities. These are documented and constraints on time ordering of actions are expressed as structure diagrams. Initial modeling produces a model of the application domain in terms of communicating processes that correspond to actions in the structure diagrams. The function step inserts the functions into the model as executable operations that produce required outputs. Finally, the purpose of system timing is to specify constraints on the implementation.

2.2.1.1.4 Software Quality Deployment

Software Quality Deployment (SQD) is a quality system for client satisfaction. It investigates needs and requirements of the client with the aim to improve the software system and the development process. SQD is driven by “the voice of the customer.” The method uses a series of matrices to deploy customer and quality requirements [Akao 1990]. These deployments are a structured way of dealing with a special concern at a detailed level, and provide a basis for linking these concerns into an integrated framework. SQD identifies, explicitly states, and prioritizes requirements. The SQD process builds a series of matrices correlating user needs and technical requirements. The matrices comprise of the client characteristics/client segments matrix, the client segments/client requirements matrix, the client requirements/technical requirements matrix, and technical requirements/entities-and-processes matrix.

Firstly, customers and users are identified and classified in client segments. The problem is outlined and the development horizon is defined. Then primary needs of the customer are documented and “refined.” Information about needs can come from various sources, e.g. customer opinion surveys and market research data. The requirements are then used in client segments to understand which segments value which requirements and to what extent. This matrix indicates whether defined

requirements cover client needs and depicts conflicts between needs and requirements. It also provides an elaborate method to prioritize requirements on the basis of their importance to the customer.

2.2.1.1.5 Soft Systems Methodology

Soft Systems Methodology (SSM) views Requirements Engineering as a recurrent cycle that aims at improving the problem situation [Checkland 1989]. It applies systems thinking to explore the problem situation and to specify the software systems. SSM assumes that different views of the application domain exist and that these views must be considered while generating requirements. The method compares perceptions of the problem with models of its application domain. Initiated by this comparison, assumptions about the problem are drawn, challenged and tested. SSM is based on the assumption that RE proceeds via debate and that it is intrinsically participative.

Initially, the requirements engineer enters the situation considered problematic and identifies the clients for the software system to be built. A first investigation of the problem situation is carried out and documented. Next root definitions are formulated. These root definitions specify the purpose of the system in terms of transformation processes. In constructing root definitions one has to consider the mnemonic CATWOE, where the C stands for customer as the victim or beneficiary of T, the transformation process. A denotes the actors who perform the activities of the system, whereas O is the owner who can stop these activities, and E are the environmental constraints. W describes the worldview or Weltanschauung when formulating root definitions.

The next stage of SSM compares conceptual models with perceived reality using informal discussion, formal questioning, scenario writing based on operating the model, and trying to model the problem in the same structure as the conceptual model. Finally, the comparison stage uses the differences between models and reality to discuss changes, which are desirable and feasible concerning the worldviews specified in root definitions. Implementing these changes completes the SSM cycle.

2.2.1.1.6 Methodology Comparison

The following table lists the methodologies described above and maps them to the five generic activities (Requirements Elicitation, Requirements Specification, Requirements Analysis, Requirements Verification and Validation (V & V), and Requirements Management) spanning the requirements engineering process.

<i>No.</i>	<i>Methodology</i>	<i>Characteristic</i>	<i>Requirements Engineering Process Coverage</i>
1	CORE – Controlled Requirements Expression	Viewpoint oriented analysis activities	Requirements Analysis Requirements Elicitation
2	Issue-based Information Systems	Tracking Issues and propositions thereof	Requirements Elicitation Requirements Analysis
3	Jackson System Development	Formalizes Functional Specification development in a 5-step process from initial requirements	Requirements Specification (Structure Diagrams) Requirements Analysis
4	Software Quality Deployment	Using a series of matrices, phases and deliverables of the RE process are mapped and traced	Requirements Elicitation Requirements Analysis Requirements V & V
5	Soft Systems Methodology	Represents requirements in the form of <i>root definitions</i> , built as per the acronym <i>CATWOE</i>	Requirements Elicitation Requirements Analysis Requirements Specification

Table 2.1: Requirements Engineering Methodologies

CORE and IBIS, both advocate a viewpoint-oriented approach to capturing requirements. The collection of information in the form of viewpoints, actions and inputs helps the information structuring process. However, CORE does not offer an elaborate analysis and verification & validation methodology. Like CORE, IBIS is also data-oriented and focuses on articulation and resolution of viewpoints, to decide on those that will be incorporated. This methodology is especially useful for Requirements Elicitation and Requirements Analysis. However, no means or guidelines are suggested for specification. Moreover, *ad hoc* viewpoint generation may result in a lot of information. In the absence of a management principle, extracting relevant information will be a difficult task. An elicitation issue with viewpoint oriented approaches is to maintain agreement between all stakeholders. Convincing one stakeholder of another's viewpoint can be a difficult task.

Jackson System Development, as a methodology, is probably more suited to modeling activities, once they are documented, as opposed to eliciting requirements from customers. Formalizing the modeling process aids requirements analysis, and to some extent verification. However, JSD does not specify any explicit methodology or guidelines for conducting requirements V & V. On the other hand, SQD provides elaborate guidelines to elicit and "refine" requirements. They identify the existence of client needs, and their transition to technical requirements thereof. A novel methodology entails verification and validation, by drawing traceability matrices. These ensure that all needs are incorporated. However, no guidelines are included for requirements specification. Moreover, a significant disadvantage of this approach, as pointed out by [Hofmann 1993], is that the method offers no "stopping" condition for the requirements engineering process. It is the responsibility of the people who apply the method to decide when the requirements are completely specified.

Lastly, SSM provides an insight into problem identification – a crucial and oft ignored aspect of the requirements engineering, which deals with the understanding of the problem, before any requirements are elicited. This distinction between the 'problem space' and the 'solution space' provides process abstraction and distinctly divides the entire process. Requirements Elicitation is initiated via the formulation of root definitions. There is, however, no provision for requirements V & V activities.

2.2.1.2 Models and Frameworks

Models and Frameworks provide a ‘skeleton’ – an overarching structure that guides the process progression. This section presents models and frameworks that have been inspirational to the development of the Synergistic Requirements Model.

2.2.1.2.1 RGM

The RGM proposes a refinement to the conventional requirements generation process, by providing a structured framework that “first distinguishes between requirements generation and requirements analysis, and then recognizes distinctively different activities within requirements generation, i.e. indoctrination, preparation, elicitation and evaluation” (Figure 2.1) [Arthur 1999].

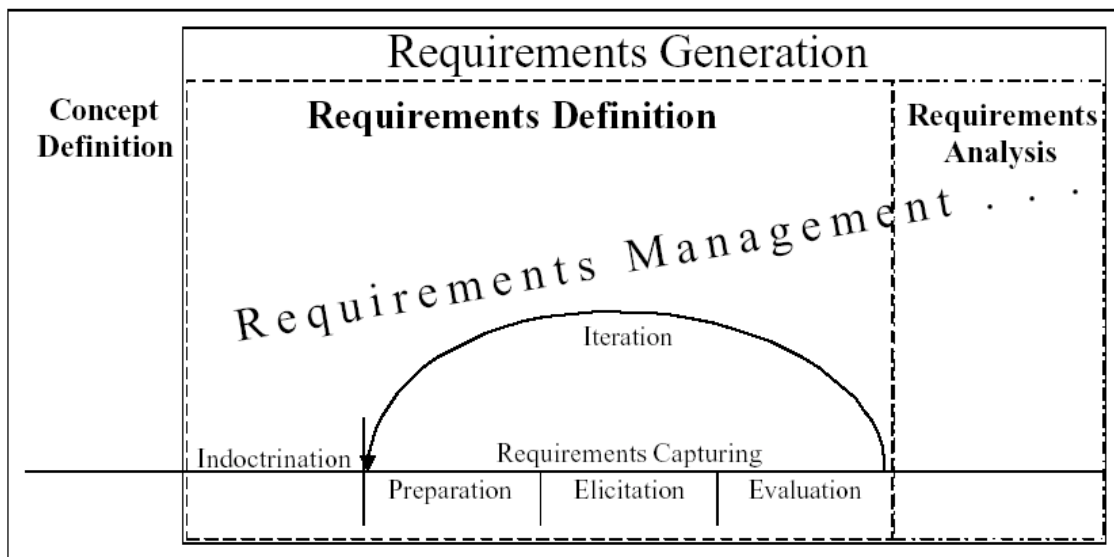


Figure 2.1: The Requirements Generation Model (RGM)

The latter three activities are embodied within an iterative process focusing on requirements capturing. The RGM provides guidelines and enforces protocols to help structure its attendant activities. Guidelines are suggestions or recommendations that offer support by serving in an advisory capacity. Protocols are rules that establish boundaries through pre-defined constraints.

The RGM provides a means of structuring the requirements capturing process by providing a distinctive framework that guides the activities through which the

customer and requirements engineer define requirements. It also incorporates a monitoring methodology that ensures all requirements elicitation activities follow proper procedures. For more details, the reader is advised to refer to [Arthur 1999].

2.2.1.2.2 *Requirements Triage*

“Knowing what not to build is as important as knowing what to build”

-Al Davis in [Davis 1999]

In his paper, Davis [Davis 1999] claims that the “software product planning” helps elicit the “right” product features, which is imperative for a development effort to be successful. In this spirit, the Requirements Triage advocates the consideration of factors such as price, market, cost, time-to-market, market size, feature mix, market penetration, revenues, profits and return on investment as a key to project success.

Software Product Planning fits between problem analysis and requirements specification and includes the activities of Risk Analysis, Cost & Schedule Estimation, Price Analysis, Market Analysis and a “Feature Triage,” as depicted in Figure 2.2.

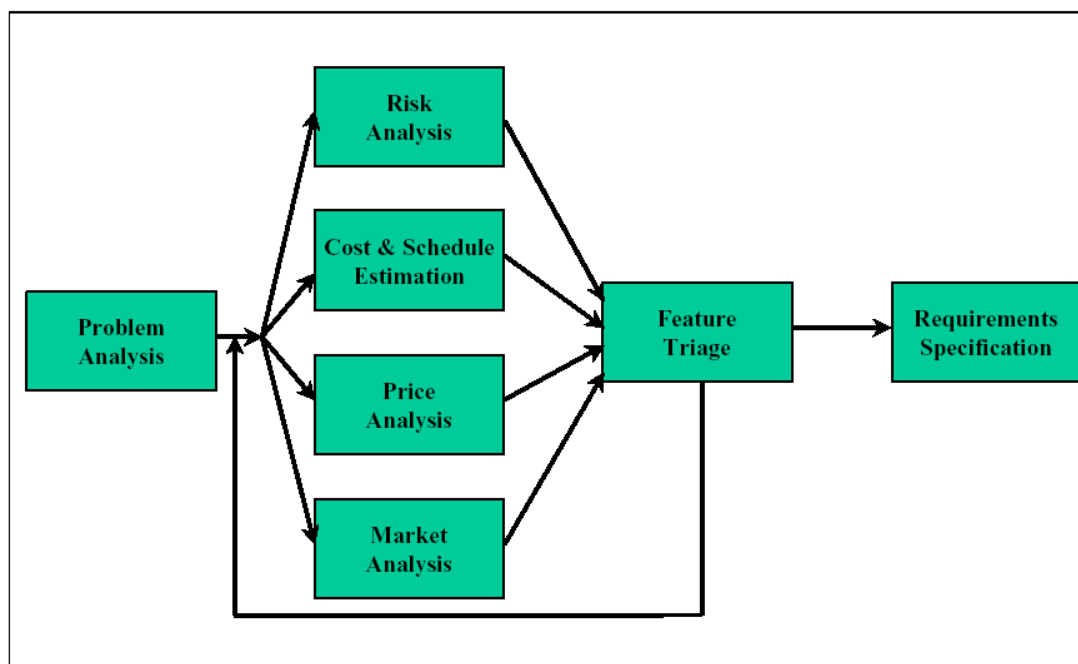


Figure 2.2: The Requirements Triage

After an initial selection of capabilities to be included in the product, the selection needs to be verified accounting for development, sales, marketing and finance perspectives. If any of these perspectives indicate a losing situation, then it can be fathomed that the company will lose, resulting in a possible failure of the project.

The process commences by eliciting features. Based on an elaborate set of guidelines (refer [Davis 1999] for details), the features are categorized and prioritized based on the relative importance, cost and risk. Next, a decision is made about the features that are worth including in the final specifications, and the ones excluded are deferred to later releases. This is done by simply “drawing a line” among the sorted and prioritized list of features, so that the sum of the efforts required for the features above the line is less than or equal to the tentative development budget. If an acceptable situation is encountered, then software product planning is accomplished. On the other hand in case something does not “fit in,” either one of the criterion may be adjusted to find a satisfactory balance.

2.2.1.2.3 Knowledge-Level Process Model

The Knowledge-Level process model provides abstraction levels to different constituents of the requirements generation process (Section 2.1). The process composition of requirements engineering, as per the Knowledge-Level process model, is depicted in Figure 2.3 overleaf.

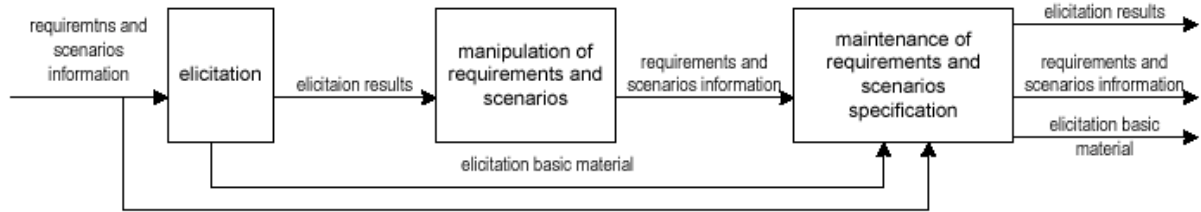


Figure 2.3(a): Model Structure

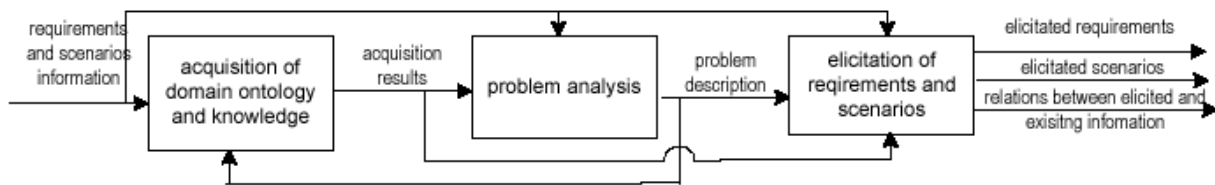


Figure 2.3(b): Processes in elicitation and their artifacts

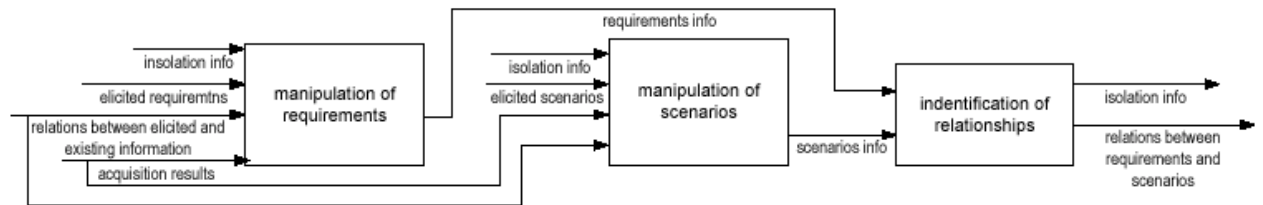


Figure 2.3(c): Processes in manipulation of requirements and scenarios and their artifacts

Figure 2.3: The Knowledge-Level Process Model

According to the composition [Herlea 1999], the “elicitation” process provides initial problem descriptions, requirements and scenarios elicited from stakeholders, as well as domain knowledge. The “manipulation of requirements and scenarios” process resolves ambiguous, inconsistent requirements and scenarios not supported by needs. This process reformulates requirements from informal requirements and scenarios, to more structured semi-formal requirements and scenarios. It also provides relationships amongst requirements and scenarios. The “maintenance” process of requirements and scenarios specification is used to store and retrieve the documents in which the information requirements and scenarios are described, including information on traceability.

The Knowledge-Level process model provides a useful approach to process modeling – that of ‘separation of concerns.’ Every process can be treated separately

and be decomposed into its constituent sub-processes or activities, facilitating a deeper understanding of the process as a whole.

2.2.1.2.4 The Win-Win Spiral Model

Boehm [Boehm 1988] reiterates the need for software process models to emphasize the need for continuous collaborative involvement of stakeholders in the definition and development stages.

According to the Win-Win Spiral Model for requirements generation, activities are performed in a spiral-like fashion, as depicted in Figure 2.4 below.

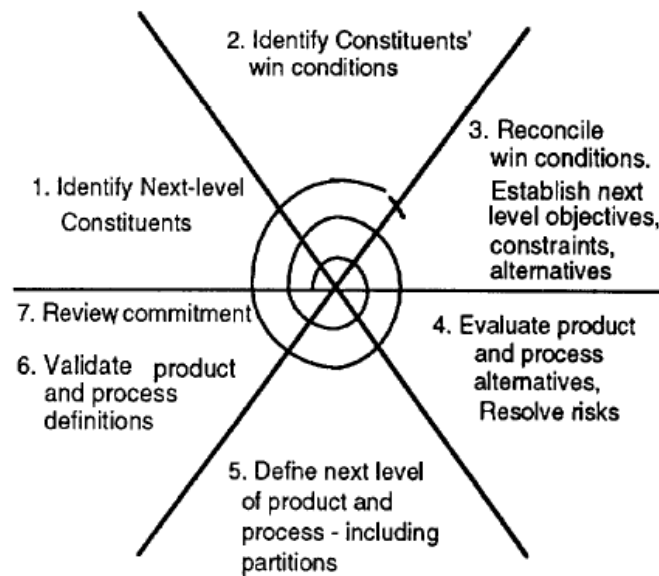


Figure 2.4: The Win-Win Spiral Model, from [Boehm 1988]

Following are the main constituent processes of the Win-Win Spiral Model:

Determine Objectives. Identify process constituents and establish system boundaries and external interfaces

Determine Constraints. Determine the conditions under which the system would produce a win-lose or lose-lose outcome

Identify and Evaluate Alternatives. Solicit suggestions from sources. Evaluate them with respect to constituent's win conditions. Analyze, assess and resolve win-lose or lose-lose risks

Record Commitments. Identify and record problematic areas that may require attention in the future stages of the development life cycle.

Cycle Through the Spiral. Resolve risks, prototype, accumulate responses and perform another iteration through the spiral, if needed.

2.2.1.2.5 Model Comparison

<i>No.</i>	<i>Model</i>	<i>Characteristic</i>	<i>Requirements Engineering Process Coverage</i>
1	Requirements Generation Model (RGM)	Structures the process of capturing requirements from the customer	Requirements Elicitation Requirements Verification
2	Requirements Triage	Extensive Requirements Analysis, based on “right” product features	Requirements Analysis Requirements V & V
3	Knowledge-level Process Model	“Separations of Concerns” aid process abstraction	Requirements Elicitation Requirements Management
4	Win-Win Spiral Model	Continuous, Collaborative development, refining requirements with each cycle	Requirements Management Requirements Analysis

Table 2.2: Requirements Engineering Models and Frameworks

The Requirements Generation Model (RGM) is the only process model that explicitly identifies the five generic requirements engineering activities. The RGM provides explicit guidelines and “protocols” for the processes it encompasses. It recognizes the presence of Requirements Analysis and Requirements Management, but does not expand its list of guidelines to include these two processes. The Requirements Triage, on the other hand, incorporates an elaborate Requirements

Analysis methodology. Assessing the features and determining the impact of these on the overall development effort presents a very practical approach to requirements engineering. Evaluating the ‘fate’ of the product from multiple standpoints facilitates verification and validation, thus ensuring that the product is feasible. It however, lacks a methodology for guiding the elicitation and management processes.

The Knowledge-level Process Model advocates a scenario-based approach to requirements generation. Although, it provides elicitation guidelines and transformation of requirements and scenarios into formal specifications, the analysis and V & V components are missing. The method of process abstraction is useful to specify the details involved in any particular process or sub-process. Finally, the Win-Win spiral model, based on an iterative approach, provides novel guidelines for analysis. The original Spiral Model, proposed by Boehm [Boehm 1988], emphasized the importance of ‘risk analysis’ in the overall development process. The Win-Win spiral model applies this to the requirements engineering process. However, this process model assumes that requirements are already elicited. No elicitation and requirements specification approaches are incorporated.

2.2.1.3 *Methods*

If the framework is viewed as a skeleton of a process model, then a method can be construed as the *organs* that populate the skeletal system. In other words, the framework provides an overarching structure to the process. Each phase in a framework can be conducted by employing different methods, depending on the need of the project. In this section, we present commonly employed requirements engineering methods. Note that each method possesses advantages and disadvantages. We do not advocate the use of one method over the other; instead the selection of a method should be pre-decided by the requirements engineer, depending on the circumstances. In doing so, we provide enough flexibility for the process to be tailored according to specific needs.

ELICITATION METHODOLOGIES

2.2.1.3.1 *JAD*

Joint Application Design (JAD) was developed by IBM Canada in the late 1970s. This technique is a structured meeting approach that provides an environment for clients and software engineers to work together as a team to share information and ideas through facilitated sessions throughout the development lifecycle. Together, participants identify needs, refine requirements, explore alternatives and select appropriate implementation strategies. As a result, requirements engineers increase their business knowledge and customers get more involved with the development process. As a result of such collaboration, potential defect arising from ambiguities and misinformation can be reduced.

JAD meetings are structured and are led by a facilitator who is ideally familiar with group dynamics and techniques for managing meetings. In JAD meetings, customers explain their problems and needs. These are documented and are analyzed, ultimately culminating in a “software requirements specification.” JAD advocates the continuous involvement of all *classes* of stakeholders throughout the development process.

JAD helps reduce ambiguities in defining requirements. Since all stakeholder *classes* are involved, a feeling of joint ownership is propagated. JAD participants also share their visions about the proposed system and gain substantial knowledge from each other. However JAD does possess disadvantages too. JAD meetings are often too demanding on the schedules of participating stakeholders. Moreover, valuable time may be wasted in resolving personnel difficulties [Chin 1995].

JAD provides the much-needed structure to the elicitation process and promotes a participatory approach to system development, ensuring that the system built adheres to the needs of the customers. This minimizes or eliminates potential rework and simplifies future maintenance.

2.2.1.3.2 PD

Participatory Design (PD), often termed as the “Scandinavian approach” to systems development advocates a high degree of user involvement in the development process, one in which “workers” actively engage in designing system, which they will eventually use.

Like JAD, PD also focuses on facilitated interaction between users and developers employing “dynamic group techniques” for eliciting and refining requirements. PD differs from JAD in participant selection, team orientation, structure and development speed [Carmel 1993]. These differences are indicated, as a part of guidelines and tools, in Appendix B.

PD seeks to emphasize the social context of the workplace and promote users’ control over development issues. PD advocates the participation of users in the requirements elicitation process. According to PD, the success of the system is proportional to the degree to which the users of the system are involved in its design and development.

2.2.1.3.3 FAST

Facilitated Application Specification Techniques (FAST) advocate a team-oriented approach to requirements gathering. It encourages the creation of a joint team of customers and developers who “work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements” [Zahniser 1990]. Key factors outlining FAST are

- Meetings are held at a neutral site and are attended by representatives of the customer as well as the developer
- Facilitator conducts the meeting employing pre-decided rules of engagement
- Goals to be accomplished are: identify problem, suggest and negotiate solution approaches, and define preliminary requirements

FAST offers a way of structuring the interaction between the customer and the developer and *promotes a sense of “we” rather than “us-them”* [Goguen 1993]. It facilitates instantaneous face-to-face discussions and refinements.

ELICITATION TECHNIQUES

The following elicitation techniques are commonly employed to obtain information from stakeholders.

2.2.1.3.4 Introspection

Introspection is the first and most obvious method for trying to understand what properties a system should have in order to succeed. It amounts to “imagining what kind of system I would want if I were doing this job.” [Goguen 1993] This method can be very useful in determining the needs of the user. It may be augmented by field studies and role-playing to improve the results. However, introspection has a problem: an expert in a different field, such as Requirements Engineering, is unlikely to reflect the experience of actual users. Personal perspectives (of experts in one field) may differ from those in another field. Hence it is imperative that one or more of empirical methods, such as those described below supplement introspection.

2.2.1.3.5 Questionnaires and Interviews

Interviews and Questionnaires are a popular means of eliciting information about work practices and needs of users, specifically when user groups are too large to engage in facilitation sessions.

Interviews – Interviewing is appropriate for obtaining information about present settings and the environment in which the new system is to be deployed. It provides opinions and viewpoints that can be instrumental in deciding the features to be incorporated into the system. It is important to select a representative group of users for conducting interviews. In other words, it is equally important to include the upper management, managerial staff and representative users, in order to elicit a complete perspective of viewpoints and opinions. Apart from individual interviews, group interviews may be conducted to seek information about present work and problems.

Questionnaires – Questionnaires have the benefit of appearing scientific, as claims can be supplemented by statistical information. In order to collect this statistically relevant information, questionnaires can be of varying types. In one case, they can comprise of questions, the answers of which can be elicited in the

form of: *very difficult, quite difficult, easy, very easy*...In another case, it can contain open questions seeking viewpoints or ideas; e.g. “*Elucidate the biggest problems in your daily work,*” or “*List suggestions for better IT support for your daily work.*” It may be difficult, however, to resolve ambiguities resulting from any of the questions, since there is no real physical interaction between respondents and the requirements engineer. Thus it is important to conduct pilot studies and resolve ambiguities before circulating the questionnaires.

Focus Groups – The focus group is a kind of group interview, but is more structured. It starts with a phase where participants come up with problems in the current way of doing things. Next, participants try to imagine an ideal way of doing things. The group also tries to explain why the ideas are good. This helps formulate goals and requirements for the new system [Lauesen 2002].

2.2.1.3.6 Protocol Analysis

Protocol Analysis asks a subject to engage in some task and concurrently talk aloud, explaining his/her thought process. Protocol Analysis is not a reliable guide to what subjects are thinking, and is open to serious misinterpretation by analysts, who can choose a small sample of protocols and develop misconceptions about an issue.

A variant of Protocol Analysis is Task Demonstration. The users are asked to demonstrate how they perform routine tasks. In many cases, users are unable to verbalize the jobs they perform, but are able to demonstrate the sequence of actions performed. This provides an insight into rare, critical tasks that may be overlooked while conducting interviews or circulating questionnaires.

2.2.1.3.7 Survey of Existing Systems and Documents

Existing systems provide invaluable insight into the procedure of operations and shortcomings encountered in present systems. These shortcomings help elicit requirements and features that must be incorporated into the new system. In the event of the absence of an old, existing system, it is useful to analyze organizational documents, seeking pertinent activities. These activities may be indicative of current

practices and can better inform the requirements engineer about the features to be considered while drafting the SRS.

To effectuate the study, the requirements engineer can study existing documents such as forms, letter files, computer logs and documentation of the existing system. Screen dumps from existing systems may also be printed [Lauesen 2002].

2.2.1.3.8 Ethnographical Studies (Analysis of Work Practice)

Ethnographical Studies are conducted to analyze the system while it is in use. This elicitation method provides insights into details about current practices, which otherwise are inexplicable. The requirements engineer spends time with the users, observing their daily tasks. In some cases, video cameras can be used to lengthen the period of observation. These tapes can later be reviewed with the user seeking clarifications.

Ethnographical studies can vastly improve knowledge of the current system and associated problems. It can also verify other information.

2.2.1.3.9 Prototyping and Experimentation

A prototype is a simplified version of a part of the final system. It can be in the form of a computer program, document (paper prototype), or a set of screenshots that implement/represent a part of the system requirements. The prototype can be used to assist in defining requirements, or evaluating alternatives. Examples include, determining how to obtain the required accuracy in real time performance issues, user interface suitability, user satisfaction levels, etc. The process of building the prototype also exposes and eliminates a number of ambiguities, inconsistencies and misunderstandings incorporated in the specifications.

2.2.1.3.10 Method Comparison

No.	Method	Characteristic	Requirements Engineering Process Coverage
1	JAD	Structured meeting approach for developers and customers to elicit needs and requirements	Requirements Elicitation Methodologies
2	PD	Facilitated interaction between developers and users, the latter taking a proactive approach in requirements elicitation	
3	FAST	Facilitated interaction between developers and customers employing pre-decided “rules of engagement”	
4	Introspection	“Imagining what kind of system I would want if I were doing this job”	Requirements Elicitation Techniques
5	Questionnaires and Interviews	Information collected and analyzed for a large representative set of users	
6	Protocol Analysis	Users are requested to demonstrate and explain “normal, everyday” activities	
7	Survey of Existing Systems and Documents	Provides insight into shortcomings of existing systems, thereby avoiding them in the new one	
8	Ethnographical Studies	Requirements Engineer becomes integral part of the user to study the context in which the system will be deployed	
9	Prototyping and Experimentation	A <i>rudimentary</i> version of the proposed system helps define requirements or evaluate alternatives	

Table 2.3: Requirements Elicitation Methodologies and Techniques

JAD, PD and FAST provide structure to the activities of requirements elicitation, facilitating interaction between stakeholders. Guidelines and rules are established for each of them and strictly followed, imposing the structure. Requirements Elicitation

techniques, on the other hand, are means to realize the *actual* collection of information. Some or all of the techniques specified may be incorporated, depending on the context of development.

2.2.2 A Synergistic Approach? (Holistic support for Requirements Engineering)

The methodologies, models and frameworks discussed in this chapter have a common goal: developing quality software requirements. They offer variegated approaches, any of which may be suitable for the RE process, depending on the project. However, with so many approaches available for improving the requirements generation process, we need to raise the question: why do 72% of projects still continue to fail due to requirements-related issues [Standish 1995].

We find the answer in studying Tables 2.1 through 2.3. Every approach described therein lacks a *holistic* approach to the RE process. They all describe activities to be performed in one, two or three phases of the process. It can be argued that a combination of these approaches will ensure coverage of the whole process, but it is inherently difficult to combine them in an *ad hoc* manner. Without any guiding structure, a combination of approaches will result in confusion among developers and customers. Hence, a *guiding* structure, or a framework is required to provide guidance and supplementary tools ensuring smooth transition between phases.

A guiding structure, incorporating any of the approaches, must provide a visible breakdown of the individual tasks in generating requirements. Furthermore, for high quality requirements, the selected approach(es) must be integrated within a framework dictating the sequence of activities to be followed. However, a framework and structure are not the only (missing) elements needed for an improved requirements generation approach. These must be supplemented with an integrated set of tools, artifacts and guidelines that assist the requirements engineer in assimilating requirements.

2.3 Requirements Generation Environments

Over the recent years, the term “environments” have been associated with development platforms, which provide utilities, such as editors, compilers, debuggers, etc. to write, execute and test programs. Environments such as Microsoft’s Visual Studio

provide elaborate CASE tools that simplify the process of programming and debugging. Providing visual representations of memory locations and contents of variables aids the programmer in determining the flow of execution. The programmer can “step through” the code to pinpoint the source of the problem. Other CASE tools, such as Rational Purify[®], help improve efficiency of code by diagnosing memory leaks, thereby preventing subtle errors that if gone unnoticed could cause problems in the overall functioning of the system. Similarly, design environments, such as Rational Rose[®], provide tools that aid representation of “design entities.” Program “test beds” automate the testing process, by automating the development of test cases, testing normal operations and boundary conditions.

Amidst the *hoopla* of these revolutionary CASE tools, the software development community has conveniently chosen to ignore the development of CASE tools supporting the requirements engineering process, even though they acknowledge the importance of the process. Existing “environments” for Requirements Engineering are few and far between. Moreover, they are all based on a formal approach to gathering requirements. The primary disadvantage of a formal approach, apart from the learning gap, is the lack of involvement of the customer. On the other hand, formal approaches minimize ambiguities that are prevalent in natural languages.

In what follows, we describe two *so-called* Requirements Engineering Environments, and point to the insufficiency thereof. It will be realized that the environments enunciated below, are quintessentially elaborate “specification” tools. Neither of them provides guidelines as to how to gather requirements. They simply assume that requirements already exist, and discuss suitable methods of representing them, with the goal of eliminating ambiguity and improving overall quality of the SRS.

2.3.1 CARD: A Software Requirements Definition Environment

CARD is based on a requirements model that enables a “requirements definer” to produce an SRS possessing good quality attributes, such as, correctness, testability, traceability, feasibility and usability [Ohnishi 1995]. The model supports requirements analysis, describing requirements, SRS verification, SRS “execution” and preliminary software design.

The CARD environment consists of three systems: Requirements Description System, Requirements Description Revising System, and Design Supporting System. CARD is based on the observation that every requirement is a collection of nouns and verbs as its components, and there exists a relation among these components. It represents the components in the form of Requirements Frames. These Requirements Frames, in turn define the structure of a “concept.” A concept has an agent, source and goal. The agent corresponds to data, which is transferred from the source to the goal. In effect, a concept is derived for every requirement providing a pictorial, formal representation. This representation is then expressed in a formal language that is subjected to verification. The representation in the formal language is also amenable to retrieval, much like the concept of searching through a database of requirements statements. Finally, an “executer” is capable of generating a rapid prototype indicating information flows and transformation of data through the system.

CARD does contain elements of an integrated environment, but does not offer support for actually gathering requirements. For any development effort, CARD may be deployed once the requirements are obtained, for the purposes of verification and generation of prototypes. However, any formal representation of requirements is hard to comprehend for customers.

To summarize, CARD offers an innovative platform to reduce ambiguity, commonly found in natural language specifications and offers built-in verification tools. A striking feature is the capability to generate prototypes indicating information flows. This capability can aid the verification process with the customer, ensuring that the system will perform as per expectations.

2.3.2 READS: A Requirements Engineering Tool

READS supports the tasks of requirements “discovery”, analysis and decomposition. It comprises of a requirements browser, a requirements inspection window, a document display window and an allocation browser. The requirements inspection window contains requirements undergoing creation, review, editing or decomposition. The goal of this representation is the development of a set of designable requirements that are precise, unambiguous and testable. The document display window support requirements

discovery and extraction. This innovative feature of READS extracts requirements from large documents and stores them into a database automatically. This is facilitated employing a user specified or “canned regular expression” to identify requirements statements and tokens. READS also provides a document versioning tools that link unchanged requirements to their new location in the revision [Smith 1993].

READS offers innovative tools, such as those described above to augment the requirements engineering process, enhancing organization and clarity in the specification. But on careful introspection, we realize that terming READS, as an “environment” would not be appropriate. Our understanding of the term “environment” entails an amalgamation of tools, artifacts, guidelines and protocols, presided by an overarching process that dictates the circumstances under which these tools, artifacts, guidelines and protocols are to be deployed. This presiding process is found to be absent in READS.

It is our assertion that tools such as those offered by READS, will only be realized to their complete potential if they are used in the *correct* phase of the process. Moreover, it would be useful to integrate tools such as these in an “application,” or an “environment,” that would automatically present the tools to the requirements engineer when the need to do so arises.

2.4 Development of the Requirements Generation Model

This section is a prelude to Chapter 3 and 4, which describes Synergistic Requirements Generation Model and Tool respectively. Identifying the insufficiencies of the approaches and the environments presented in Section 2.2 and 2.3, we have conceived a model and have developed an environment based on this model. Sub-section 2.4.1 describes key issues that were considered while conceiving the requirements generation model.

2.4.1 Key Issues

We enumerate certain key issues that are addressed during the conception of the Synergistic Process Model for Requirements Generation and subsequently incorporated in *itSRGM*.

2.4.1.1 Identifying the *real* problem

Activities in requirements engineering are diverse in nature and approach. Each of the activities present different issues to those involved in executing them. According to Groener [Groener 2002], 80-90% of all effort on requirements is spent on analyzing and improving what has been received from the customer, while only the remaining 10% of the effort is expended in actually gathering requirements from the customer. Therefore, very little time is devoted to this all-important activity of interacting with the customer to determine and correctly capture not only their requirements but also their intentions. Subsequently, it is not surprising that in the end we develop a product that meets all the documented requirements, yet the customer does not use the product because the intentions and needs were not met.

Current approaches lack the important activities of supporting the *elicitation* of customer needs. Most approaches seek information from the solution perspective, i.e. what has to be built, without considering whether the solution that has been devised (for which requirements are being elicited) is right or wrong. It is important to determine whether the problem expressed by the customers is the real problem or not. Extensive investigation should be conducted to achieve confidence, proceeding into the software requirements gathering stage.

2.4.1.2 Communication (A *Participatory* Approach)

The need for structured communication among all stakeholders is paramount to ensure that the system built will be the system used. Requirements gathering should be a participatory approach that involves customers in all the aspects of development, including gathering, analysis and verification. Communication takes place at different levels. Initially, the customer approaches the development team with a problem. The development team then seeks to identify all concerned stakeholders, study existing systems and work practices, with the intent of recognizing the problem at hand. Various interaction methods such as those explained in Section 2.2 can be employed to determine the problem. Once identified, requirements are elicited for the problem. The requirements are then analyzed based on the overall system context and constraints imposed by the customer. Finally, these requirements are verified and

validated with the customer, to ensure that they are representative of the customer needs. In doing so, system development can proceed into the design and implementation stage with a high degree of confidence.

2.4.1.3 Integration

Many tools and approaches exist to aid the requirements engineering process and improve the quality of requirements (Section 2.2, 2.3 and Appendix B). They all offer variegated support, helping stakeholders through the process. The only component missing from the plethora of tools available is an “*Integrator*.” In our case, an Integrator is an overarching structure that guides, schedules, manages and invokes pertinent tools at the correct instance. In other words, an Integrator can be regarded as a process model comprising of tools, artifacts, guidelines and protocols. Most importantly, the Integrator must also provide some representation to guide the requirements engineer through the process.

2.4.1.4 ‘Interwoven’ Verification and Validation

Software Requirements Verification and Validation is an aid in determining that the software requirements are specified correctly and completely and are traceable to the initial inputs. Software Verification examines the products of each development activity to determine if the software development output meets the requirements (or constraints) established at the beginning of the activity. Software Validation, typically performed at the end of the software development lifecycle, ensures that that software is a correct implementation of the (system) requirements established at the start of the development process.

More so than in any other phase of software development, it is imperative that the requirements specification be verified with all stakeholders as and when they are developed and also validated against the customer’s needs. This will not only ensure that the product is being built correctly, but will address the issue of building the right product. Once again, we conjecture that a participatory approach will aid the process of constant verification and validation. As stated before, the requirements produced in the specification are validated against the needs. The needs generated earlier in the

requirements engineering process are to be validated against the problem identified. The problem, in turn, will be traced back to the initial idea, the con-ops document or the high level requirements obtained from Systems Engineering. Thus, interweaving verification and validation in the proposed process model ensures that the customer intent is always captured, and the development effort is always proceeding in the *right direction*.

2.4.1.5 A Quality Approach

The requirements engineering process initiates the formal development effort. The activities performed in this process result in the culmination of the SRS, which guides and dictates the development activities performed henceforth. Merely stating that the SRS should adhere to quality standards would be an understatement. The requirements listed must be clear, unambiguous and must be in adherence to all the needs listed initially by stakeholders. Product quality is of foremost importance to prevent any misgivings later in the development lifecycle.

We believe that steps to ensure product quality should be implicit and built-in into the process. In other words, it is hard to ‘inject’ quality into ‘specified’ requirements, unless it is instilled right from the start. The requirements generation process should ensure that (strict) quality guidelines are adhered to during the generation process.

In the next chapter, we introduce a requirements generation model that quintessentially represents a participatory approach to requirements generation, and to a large extent, addresses the key issues represented above. The model provides a guiding structure to the activities performed in the requirements engineering process.

Chapter 3 The Synergistic Requirements Generation Model

3. THE SYNERGISTIC REQUIREMENTS GENERATION MODEL

After having motivated the need for a holistic process for requirements engineering in Chapter 2, we now introduce the Synergistic MODEL for Requirements Generation, and its components.

The scope of our work, within requirements generation, pervades throughout the entire process. It is our attempt to represent a model that guides the requirements engineer, starting with generating a conceptual overview of the system, recognizing the problem at hand, eliciting requirements, specifying and analyzing them, and finally validating the requirements against previously generated documents. The approach that we advocate comprises 2 parts – (1) A Model, and (2) An Interactive Tool, implementing/codifying the model. In this chapter the primary focus is on the model. The interactive tool is discussed in Chapter 4. We begin with a brief introduction to the modeling technique adapted to represent the process model.

It is our intention that through this model, the requirements generation process will obtain the much-needed structure and will also provide the means to link together the (disconnected) approaches discussed in Chapter 2. Not only does the model dictate the sequence of activities to be performed, but also identifies tools and artifacts that aid requirements management.

Process Modeling (*“Separation of Concerns”*)

Process Modeling was crucial while devising the Synergistic Requirements Generation Model (SRGM). Representing phases in a way that is lucid and intuitive was critical to aid the overall understanding of the process. Moreover, the process must be visually represented as a complete snapshot, enabling the requirements engineer to determine the context in which he/she is working. It is also imperative that the process model provides a representation of the progression of activities performed currently and in the future. To address these concerns associated with requirements generation, we employed a software engineering modeling technique – “Separation of Concerns” [IBM

2003], for *process decomposition* to identify the low-level activities, and high-level abstractions to enable the user to abstract back to the higher-level concepts to determine the context in which he/she is working.

“*Separation of concerns* is a concept that is at the core of software engineering. It refers to the ability to identify, decompose and manipulate those parts of a software process that are relevant to a particular concern (concept, goal, purpose, etc.). These concerns are the primary motivation for organizing and decomposing software into manageable and comprehensible parts [IBM 2003].” Every process and sub-process is decomposed to “low-level” activities or sub-processes, in an effort to make the overall process as *lucid* and *coherent* as possible. At the lowest level of decomposition, tools and artifacts are identified, aiding the activity/sub-process. Initially, the requirements generation process is split into 5 components/processes – Conceptual Overview, Problem Synthesis, Requirements Capturing, Requirements Analysis and Requirements Validation. In turn, each of these 5 processes is further split into sub-processes and activities. For example, the requirements engineer typically starts with the Conceptual Overview process, providing the (initial) input source. This is the highest level of abstraction. The Conceptual Overview process is then decomposed into lower level activities of “Initial Customer Interaction” and “Documenting System Characteristics” (in a sequential order) producing the output of the first stage – the Problem Description document. Similarly, in this manner, the other processes are decomposed into lower level sub-processes and activities. Providing process decomposition helps focus on individual pieces that comprise the whole model.

We have purposely conceived the SRGM to reflect a ‘waterfall’ characteristic, implying that software requirements are generated at the start of the software development process. Furthermore, the SRGM assumes that development is occurring within the waterfall model, or a variant thereof. This was done with the intention of focusing on “Requirements Generation” as a process distinct from the other phases of the software development lifecycle.

Components of the Synergistic Requirements Generation Model

At the highest level, the Synergistic Requirements Generation Model consists of 5 processes and intermediate artifacts, exchanged between those processes, as shown in Figure 3.1. Each of these 5 processes and the artifacts they produce are explained, in detail, in the sections that follow. Templates for several of the artifacts can be located in Appendix B.

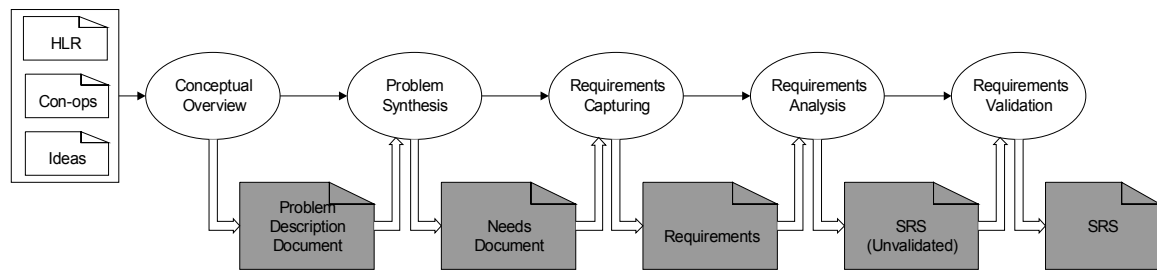


Figure 3.1: Synergistic Requirements Generation Model

3.1 Conceptual Overview

The software requirements generation process begins with an effort to generate a conceptual overview of the system. The conceptual overview enumerates the proposed system characteristics based on initial interaction with the customer.

Referring to Figure 3.2, the process commences with the customer providing the input source (Con-ops Document, High Level System Requirements, or “Ideas”) to the development team. Based on the input(s) and initial interaction, the requirements engineer documents the *primary* system characteristics, in the form of a formal, standardized document referred to as the Problem Description Document.

The Conceptual Overview is crucial since it captures the initial notion the customers possess about the proposed system or the problem that the proposed system must address. Documenting this problem is important for later validation.

Before elaborating more on this process, it is crucial to understand the difference between the inputs to this process – the Con-ops Document, High Level System Requirements and Ideas. Table 3.1 provides a description of these differences.

Input Source		
Con-ops Document	High Level System Requirements	Ideas
The customer, based on extensive analysis of the problem, formally prepares the Con-ops Document. This is presented to the developers, as a ‘work order,’ for which requirements are elicited and thereafter implemented	Often the proposed software system is a part of a larger system. In this case, the functionality of the software is already thought out in the form of ‘High Level System Requirements.’ A document enumerating the proposed functionality is presented to the developers with the intent to refine the requirements and focus on the software relevant part	The customer often has an idea of the problem and how it should be solved. These ideas must be captured and presented to the developers
These input sources differ in structure and ingenuity. The difference, as will be indicated later, is important in determining the extent of activities performed in the Problem Identification stage. In other words, the problem portrayed in the Con-ops Document or High Level System Requirements will be amenable to <i>less</i> verification, as compared to the problem identified through the customer’s ideas.		

Table 3.1: Sources of Input of the Synergistic Model for Requirements Generation

In what follows, we provide a description of the activities involved to generate the Problem Description document – the artifact produced as an outcome of the Conceptual Overview process. A tabulated overview for each of the phases has been enumerated in Appendix A. We conclude the section by providing a template for the Problem Description document.

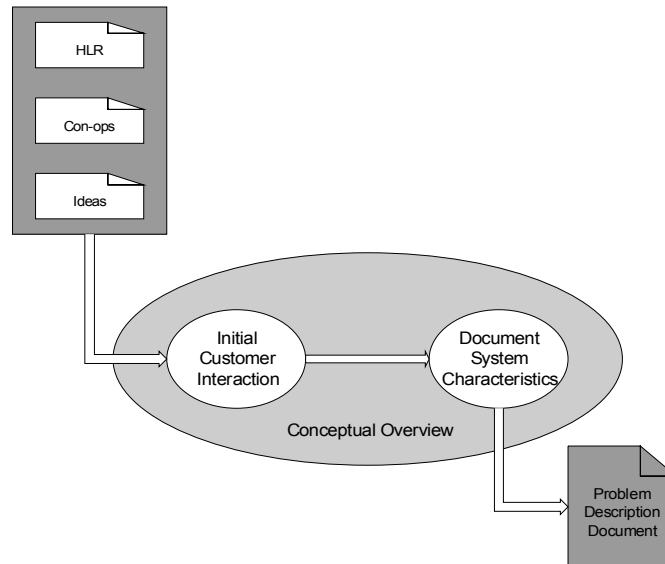


Figure 3.2: Conceptual Overview

3.1.1 Initial Customer Interaction

Whenever a new project is contracted, it is important to document the initial interaction occurring between the two parties – the customer and the developer or requirements engineer. Initial interaction results in a *portrayal* of the problem, as the customer perceives it. The development organization, in turn, can present to the customer, the infrastructure and expertise it possesses. The initial interaction culminates in a memo of understanding (a preliminary contract) between the customer and the development organization, outlining the problem and a brief description of the proposed solution. Moreover, it paves the way for further analysis.

(Note that in this interaction, the problem presented by the customer may be correct or incorrect. It can be assumed to be correct if the customer provides a Con-ops document or the High Level System Requirements; else an input via Ideas should be subject to verification. However, we do recommend that the requirements engineer verify the problem in either case – in the latter, to determine and verify the real problem, while in the former to gain a deeper understanding of the problem)

3.1.2 Document System Characteristics

The initial interaction between the customer and the development organization, and preliminary observations (about the problem) made by the requirements engineer must be documented in a standardized format. This aids analysis activities performed in the next phase (Problem Synthesis) of the Synergistic Requirements Generation Model.

System Characteristics include the following information:

- The rationale for implementing and deploying the proposed system from a business standpoint
- The benefits of the proposed system, along with shortcomings of the current approach/system
- The High Level Requirements, based on a cursory understanding of the need and initial customer interaction
- The overall impact of the proposed system deployment on the business
- Recommendations, if any.

The format of the Problem Description document is presented in Table 3.2. To reiterate, the information presented in the Problem Description Document provides an initial understanding of the need and the rationale for the new system. It leads to a more informed analysis in the subsequent phases.

3.1.3 Problem Description Document (System Conceptual Overview)

The format of the Problem Description Document (System Conceptual Overview) is provided overleaf:

Introduction	Proposed System overview Terminology Organization of Document
---------------------	---

Business Need	Business Problem in context to the proposed system Business Opportunities in context to the proposed system Existing System Description Organizational Constraints Types of Users
System Justification	Business Benefits Shortcomings of Current System
System Concept	Proof of Concepts High Level Functional Requirements
Operational Scenarios	Intended Usage Description Intended User Interactions
Business Impact	Proposed system impact on organization Changes in current organizational setup
Rationale	Reasons and convictions for change Possible risks involved with deployment
Conclusions/ Recommendations	Development Team recommendations Go/No Go Decision and justification

Table 3.2: The Problem Description Document
(System Conceptual Overview)

The Problem Description document is an important document in the requirements generation process. It reflects the customer's initial understanding of the system, and is hence an invaluable resource when validating the software requirements and the final system as well.

3.2 Problem Synthesis

Problem Synthesis is the second of five processes in the Synergistic Requirements Generation Model (Figure 3.1). It deals with the process of understanding the *real* problem, assessing and documenting the users' needs. In other words, the goal is to

identify and gain a deeper understanding of the problem, its context and the environment in which the system is going to be deployed. The needs generated as a result of this process serve as a validation criteria once requirements are elicited – every requirement must be mapped to the corresponding need(s) to ensure that all users' needs have been incorporated in the requirements specification.

The sub-section that follows examines the activities involved in the Problem Synthesis process. As illustrated in Figure 3.3, the Problem Synthesis process typically starts with educating the requirements engineer about the context and the environment in which the system is to be deployed. Next, the Problem Analysis phase starts with an extensive investigation into the root cause of the problem. Once identified, this root cause is further analyzed and decomposed into its constituents. This process is referred to as Problem Decomposition. For each of the problem constituents, needs are elicited from users. These needs are finally documented and analyzed, during the Needs Generation process, and are subsequently used for eliciting requirements. Each of these processes are depicted in Figure 3.3 and described in greater detail in the following subsections.

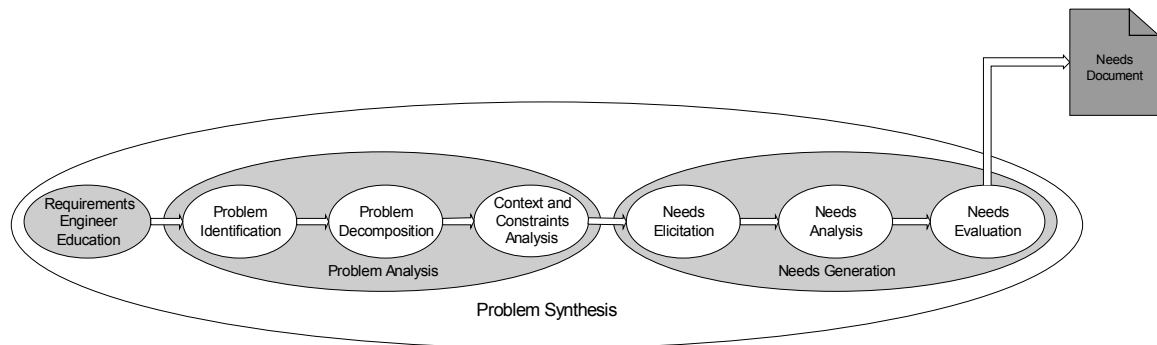


Figure 3.3: Problem Synthesis

3.2.1 Requirements Engineer Education

During the requirements engineering process, the requirements engineer is responsible for capturing requirements, organizing them, and correctly portraying the proposed system. Educating the requirements engineer is a task that must be carried

out by the customer. The customer must introduce the requirements engineer to relevant documents and stakeholders, all of which might be useful sources of information. The requirements engineer must also be introduced to the environment in which the new system will be deployed. The customer and the requirements engineer must also discuss the motivation for introducing a new system (or bring about changes to an existing system), outlining the purpose and the overall scope.

The following methods to seek information are useful.

- Interviews
- Customer-led Presentations
- Workshops
- Experiencing life as a user (ethnographical field studies)
- Studying existing documents

To summarize, the overall goal of the requirements engineer education is to obtain *preliminary* information about the current and proposed system, identify the context in which the proposed system is to be deployed and interact with stakeholders to learn the organization practices, relevant to the system.

3.2.2 Problem Analysis

Problem Analysis (Figure 3.4) focuses on ways in which the requirements engineer can understand the problem at hand, thus motivating the need for changing the current system or developing a new system altogether. The objective is to facilitate problem investigation, verification and formulation, ensuring that requirements, subsequently elicited, will represent the correct problem.

In order to describe the activities of problem analysis, it is helpful to first define what a *problem* really is. According to Gause and Weinberg [Gause 1989], “a problem can be defined as the difference between the things perceived and things as desired.” The primary intent of the requirements engineer and the development team is to solve this *anomaly*.

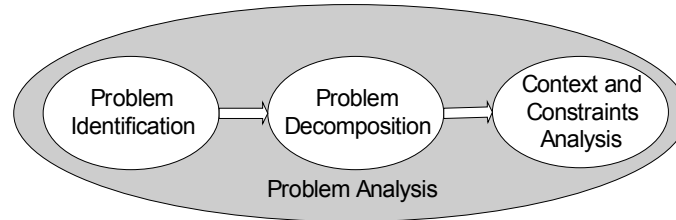


Figure 3.4: Problem Analysis

As depicted in Figure 3.4 above, the Problem Analysis process is further divided into three processes:

1. *Problem Identification*, during which the requirements engineer identifies stakeholders and conducts investigation, based on the input provided by the customer at the start of the RE process and the Problem Description document.
2. *Problem Decomposition*, during which the requirements engineer, in consultation with the stakeholders, gains a deeper understanding into the problem identified earlier. This is facilitated by using decomposition techniques to decompose the problem and identify its constituent causes.
3. *Context and Constraints Analysis*, during which the context and constraints are determined, within which the system is to be deployed. External factors having a direct or indirect impact on the proposed system are also studied.

Each of these sub-phases is discussed in greater detail in the sections that follow.

3.2.2.1 Problem Identification

Problem Identification involves activities facilitating investigation, with the intent to determine the *real* problem and its underlying causes. It is important to note that the extent of activities performed in the Problem Identification process depends on the input source type. As outlined in Table 3.1, in case the input source is the Con-ops document or High Level System requirements, it can be assumed that the problem is *correctly* identified and activities defined within this process can be conducted in a cursory manner. On the other hand, in case the input source is based on the customers' ideas, then the activities defined hence, are to be necessarily performed, within the framework specified.

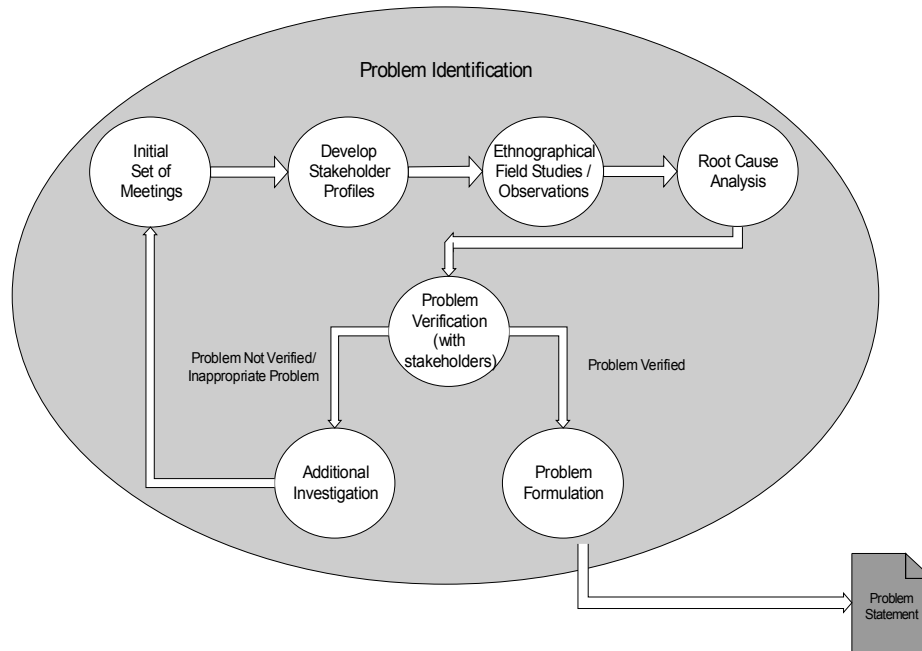


Figure 3.5: Problem Identification

As is indicated in Figure 3.5 above, Problem Identification begins with an initial set of meetings, conducted primarily to identify pertinent stakeholders and build profiles. Information such as stakeholder roles, their contact information, ‘decision-taking’ capability, and *hierarchy position*⁴ is sought. Once developed, tools⁵ assist in documenting the information.

After gaining information about the organizational setup and the roles of stakeholders, the requirements engineer must conduct ethnographical studies (Chapter 2, Section 2.2.1.3.8). This helps the requirements engineer to gain first-hand experience of the problems faced by users, the shortcomings of the current system, and the expectations of the proposed system. Collecting this information is a valuable resource for determining the root cause of the problem. Although the ethnographical approach advocates that the requirements engineer become an *integral* part of the overall organization, we do understand the time constraint this imposes on the overall process. However, we do recommend that this process be conducted earnestly. The requirements engineer can conduct interviews or solicit protocol analysis (Chapter 2, Section 2.2.1.3.6), during observations.

⁴ The hierarchy level (designation) of the stakeholder within the organizational setup is an important piece of information for the purpose of selecting meeting participants

⁵ The tools are described in Appendix B

The information collected during the ethnographical studies is then subject to *root cause analysis*. Root Cause Analysis [Leffingwell 2000] is a systematic way of uncovering the root, or the underlying cause of an identified problem, or a symptom of a problem. This technique is briefly explained with an example, adapted from [Sidky 2002].

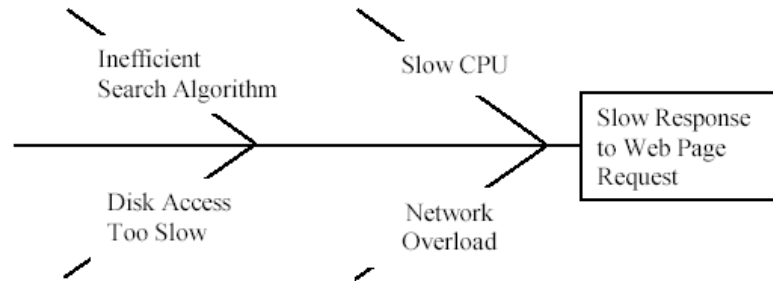


Figure 3.6: Root Cause Analysis (Fishbone Diagram)

As shown in Figure 3.6, a problem related to slow access to the organization web site, can be traced to any of the four causes. This is represented using a *fishbone diagram*. Each possible source of the problem is listed as one of the “bones” on the diagrams. The root causes can be determined in many ways. In many cases, it is a matter of simply asking the stakeholders affected by the problem. In case the problem is more serious it may be essential to conduct ethnographical and field studies iteratively. In any case, the ultimate goal is to identify the likely cause(s) for the problem.

In the spirit of the *participatory approach* advocated throughout the Synergistic Requirements Generation Model, once the problem is identified and documented, it is shared with the stakeholders. This is done with the intent of verifying the problem. In case the problem is perceived to be correct from the stakeholders’ standpoint, the problem is formally documented into a baseline document. On the other hand, an incorrect or incomplete problem must initiate another iteration of the Problem Identification process, as depicted in the overall Problem Identification process (Figure 3.5).

3.2.2.2 Problem Decomposition

Once the actual problem is identified the next step in Problem Analysis is to decompose the problem into smaller distinct elements, and to ‘refine’ their individual characteristics. The objective is to gain additional insights into the problem, and subsequently, a better understanding of the customer needs. Thus Decomposition and Refinement are activities of Problem Decomposition.

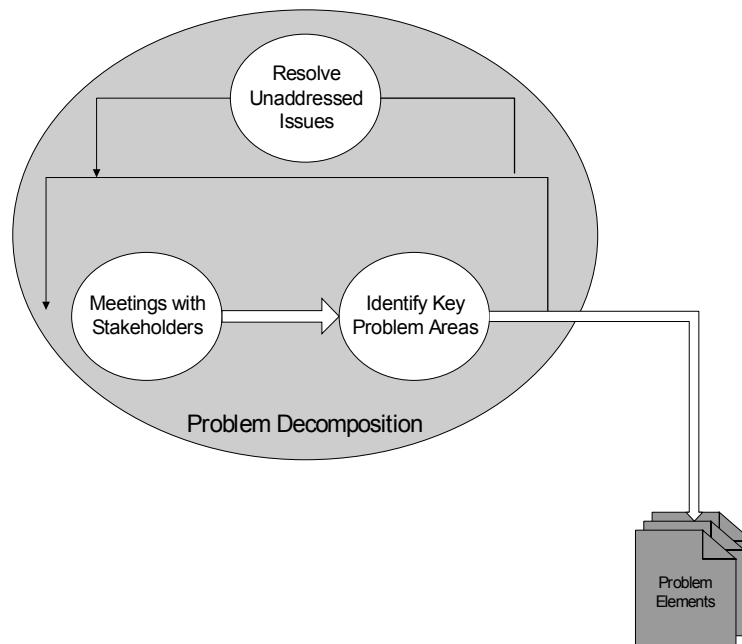


Figure 3.7: Problem Decomposition

Problem Decomposition involves a series of steps by means of which *problem elements* are obtained. These problem elements aid the generation of needs. As indicated in Figure 3.7, Problem Decomposition is an iterative process, involving (numerous) structured⁶ meetings with stakeholders. To support problem refinement, each meeting must have a focused objective and employ structured activities that support the achievement of that objective, e.g. recording decomposition components for all key problem areas.

The following approaches, all described earlier (Section 2.2.1.3), are useful during problem decomposition:

- Interviews

⁶ We introduce the concept of “structured meeting” in Section 3.3.2.1

- Preliminary System study
- Cause Effect Diagram (Root Cause Analysis).

In addition to the above approaches, one additional approach is useful, specifically to Problem Decomposition:

Pareto Analysis: Pareto Analysis is based on the Pareto Principle, which states that only a "vital few" factors are responsible for producing most of the problems: *"a great majority of problems (80%) are produced by a few key causes (20%). If we correct these few key causes, we will have a greater probability of success."*

Pareto Analysis is conducted after Root Cause Analysis, to document the different causal factors, and then decompose the problem. Collecting all probable causes (from the Fishbone diagram), and then ranking them from the most to the least importance accomplishes this. A cumulative percentage is associated with every cause, based on discussions with stakeholders. Based on the percentages obtained, certain problem elements can be diagnosed to be more important than the others, and additional resources may be committed to them. For more details⁷ on how to conduct Pareto Analysis, the reader is referred to the footnote.

3.2.2.3 Context and Constraints Analysis

Once an in-depth understanding of the problem is gained, an attempt must be made to "define" the system that can be deployed to address the problem. This implies that we must develop a scope or boundary within which the system will operate. The boundary separates the system from other systems and the real world. This helps us in identifying the inputs being processed by the system, and the corresponding outputs. Defining system boundaries is most important in cases where the software system is a part of a larger system. In this case, the interactions between the software system and the larger system can be specified through the boundaries. All interactions with the system occur via interfaces between the system and the external world.

To establish the context within which the system must operate, the data exchanged between the proposed system and other systems must be determined.

⁷ Pareto Analysis: <http://erc.msh.org/quality/pstools/pspareto.cfm>

Next, the users or actors interacting with the system must be identified. The UML notation can capture this information. Figure 3.8 outlines the activities to be performed as a part of the Context and Constraints Analysis process.

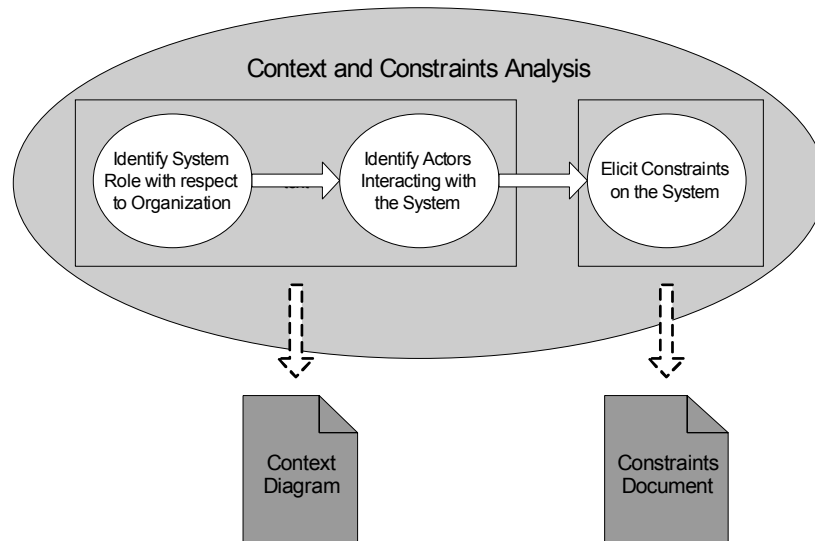


Figure 3.8: Context and Constraints Analysis

Leffingwell and Widrig [Leffingwell 2000] provide a series of useful questions that can help the requirements engineer elicit the information related to the context of the system:

- Who will supply, use, or remove information from the system?
- Who will operate the system?
- Who will perform system maintenance?
- Where and for what purpose will the system be used?
- Where does the system obtain information?
- What other external systems will interact with the system?
- What data will be transformed with the system?

With the answers to these questions in mind, the requirements engineer can now create a “system perspective” that describes the boundaries of the system, hence establishing the “context of operation,” with respect to external systems, actors, users and other interfaces.

Once the problem is identified and the solution boundaries (context) have been established, it is important to elicit the constraints within which the proposed system must operate. A constraint may be defined as “a restriction on the degree of freedom we have in providing a solution” [Leffingwell 2000]. Not accounting for a constraint can severely affect the overall development.

Sources of constraints may be variegated. Table 3.3 documents some frequent categories of constraints, adapted from [Leffingwell 2000], encountered in projects. These constraints must be elicited via continuous interaction and verification with stakeholders. Constraints related to parameters such as performance, security, operational platform, etc. also represent non-functional requirements in the SRS.

Constraint Type	Considerations
	What kind of financial or budgetary constraints is the project limited by?

Economic	Is there any procurement costs involved with the hardware/software to be deployed, if any?
	Are there any licensing issues?
Political	Are there any persons specifically affected by the system, apart from the designated stakeholders?
Technical	What kind of development platform is the system to be developed for?
	How familiar are the <i>ultimate</i> users of the proposed system with the associated technology on which the system is going to be deployed?
	Is there any technology that should be explicitly avoided during development?
	What kind of technology, if any, needs to be procured for development/deployment?
System	Will the system be required to be built from the ground up, or will it be built upon the existing one?
	What operating systems will the system support?
	What hardware infrastructure will be required to deploy the system?
Environmental	Are there any concerns regarding security?
	Are there any specific standards that must be adhered to?
Schedule	Is the schedule defined?
	Are there any specific milestones that the organization wishes to document/review?

Table 3.3: Potential System Constraints

3.2.3 Needs Generation

The output of the Problem Analysis phase includes the problem statement, problem elements and all necessary clarifications (in the form of Context Diagrams and

Constraints Documents). The objective of the Needs Generation process is to generate a set of customer needs, in consultation with the customers and users. Collectively, the problem statement, the problem elements, the Context and Constraints document and the customer needs define the *problem space*. This problem space defines what the customer needs.

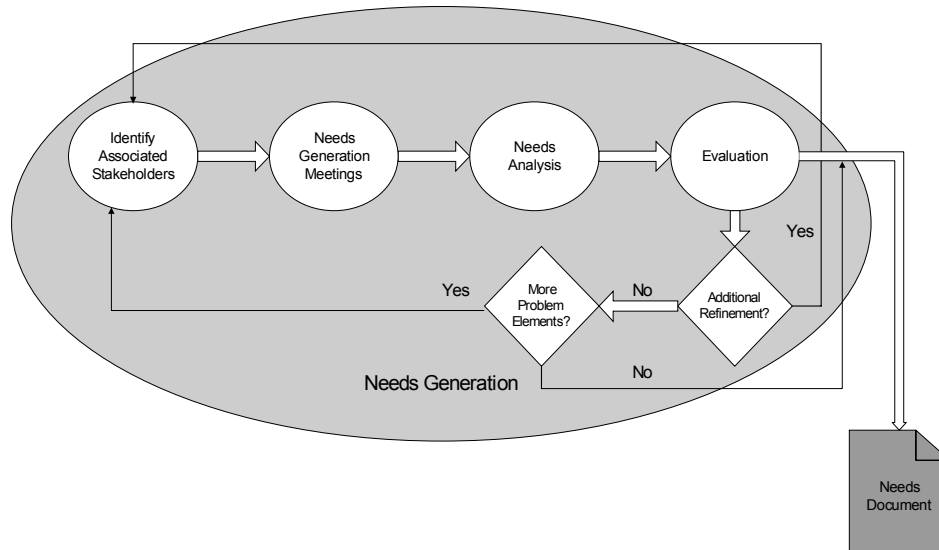


Figure 3.9: Needs Generation

The subsequent sub-sections elaborate the Needs Generation process. Once elicited, the needs are analyzed and evaluated with respect to the problem elements. In other words, every problem element can expect to be associated with one or more needs. These needs are used as an input to the requirements capturing activities. It is here that each need is translated into one or more (software) system requirement. In effect, the set of elicited requirements define the solution space – that is, they define *what* the system shall do. The Needs Generation process is depicted in Figure 3.9. The iterative structure of the needs generation process is borrowed from the RGM [Arthur 1999].

3.2.3.1 Needs Elicitation

The actual needs generation meetings are preceded by the preparation involved in conducting these meetings. The preparation is as per the “Structured

Meeting Approach.” A discussion of this approach is deferred until Section 3.3.2.1.

For every problem element devised in the Problem Analysis phase, associated stakeholders are consulted to build a set of needs.

Table 3.3 describes some subtle differences between “needs” and “requirements.”

No.	Needs	Requirements
1	Needs capture the problem (problem space)	Requirements represent the solution (solution space)
2	Needs lack formal representation structure	Requirements must be represented formally, as per the format of the SRS
3	Needs are derived from the problem elements devised in the Problem Analysis phase	Requirements are derived from needs elicited in the Needs Generation phase
4	Needs are approved based on the schedule and budget constraining the overall development effort	Requirements, once included in the SRS imply a binding contract between the customers and developers

Table 3.4: Needs versus Requirements

With these differences in mind, the requirements engineer must elicit information from stakeholders. Each problem element must have associated needs. Figure 3.10 overleaf summarizes the problem analysis processes – starting with Problem Identification, followed by Decomposition, and leading the Needs Generation. Certain useful techniques for conducting each are also enumerated.

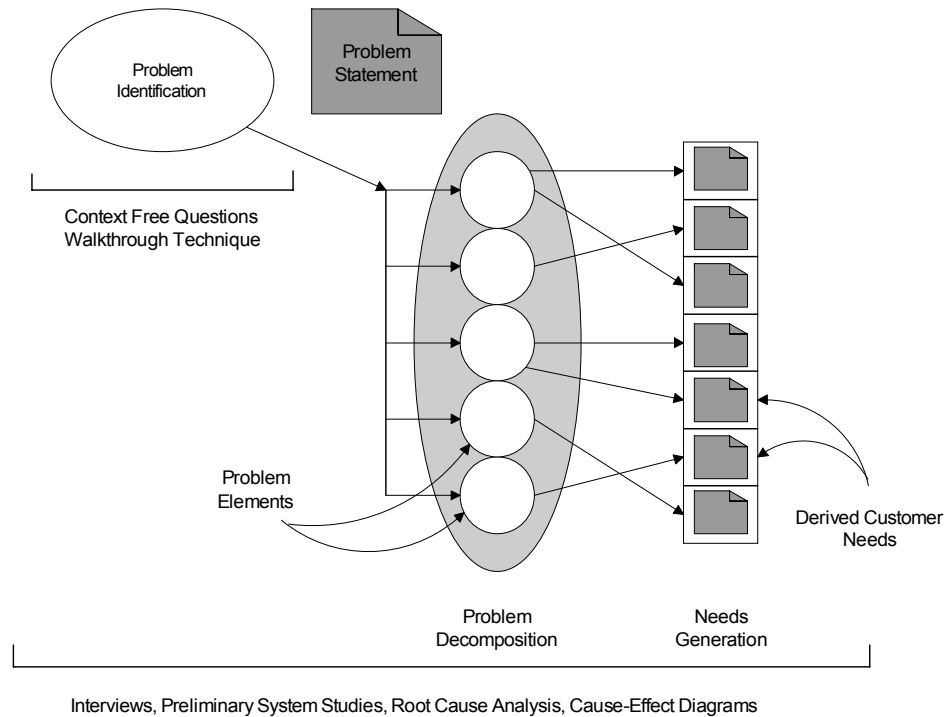


Figure 3.10: Problem Analysis (Methods)

3.2.3.2 Needs Analysis

Once elicited and documented, the needs are analyzed from the perspective of converting them to requirements. This particular phase in the Needs Generation process represents the *transition* from the “problem space” to the “solution space.”

The needs generated from the elicitation meeting are organized, related and analyzed with respect to the constraints and context, established in the Problem Analysis phase. The following techniques [Leffingwell 2000] are useful for analysis.

- a. Convergence – Once the needs are generated, they must be organized and similar ideas must be grouped. Related needs may be grouped and analyzed collectively. For example, the following groups, categorizing and converging needs may be formed:
 - New Features
 - Performance Issues
 - Enhancement to the current system

- Needs related to the user interface

Alternately, they may also be united on the basis of other parameters, such as functionality of the system or needs related to system components.

- b. Prioritization – The needs generated and categorized, as described above, must be prioritized. More often than not, the features implemented in final system are subject to financial and schedule constraints. Thus, the needs (and requirements) must be prioritized to decide upon the most important features to be incorporated. It is imperative that prioritization of the needs be done in the presence of the customer to discuss the most important features from their standpoint.

A useful approach for prioritization is during the elicitation meeting; all participants are asked to prioritize the needs. A justification is sought for assigning the ranks. Once prioritized, the requirements engineer assigns ranks to every need, based on the mean priority given by all participants. The priorities, assigned in this manner, are discussed with pertinent stakeholders during the next elicitation meeting.

- c. Conflict Resolution – Often, the needs generated during a meeting may conflict with one another. In this case, the priority of the needs must be accounted for, to decide which need is incorporated. In the case of similar priorities, the conflicting needs must be discussed with the stakeholders, weighing the pros and cons of each and reaching a decision about the one to be included. Keeping the resolution simple enables conflicts to be sorted out prior to eliciting actual requirements.

3.2.3.3 Needs Evaluation

Evaluation is conducted to determine the “reference” level, separating the needs to be incorporated from the ones that are to be left out or postponed for a future release. This “reference” level is determined based on the schedule and budgetary restrictions, in consultation with all stakeholders.

Needs are also evaluated to verify the right information and decide whether another iteration is required for the particular problem element. Iteration(s) may

be conducted to elicit additional needs or to resolve an ambiguous issue. Once the stakeholders are convinced that sufficient needs are generated, we proceed to the next phase involving the capturing of software requirements.

3.3 Requirements Capturing

Requirements Capturing is the *first* phase that elicits information about the system, seeking the *solution* perspective. The activities conducted previously represent the problem space, for which the solution has to be devised. Requirements Capturing culminates in the generation of software requirements of the proposed system. It is preceded with an optional process entailing the indoctrination of the customer. The requirements elicitation meeting follows customer education. The requirements elicited during the meeting are examined and subjected to a “Local Analysis” activity – another *novelty* of the Synergistic Requirements Generation Model. Finally, Requirements Evaluation ensures that the requirements adhere to pre-decided quality standards.

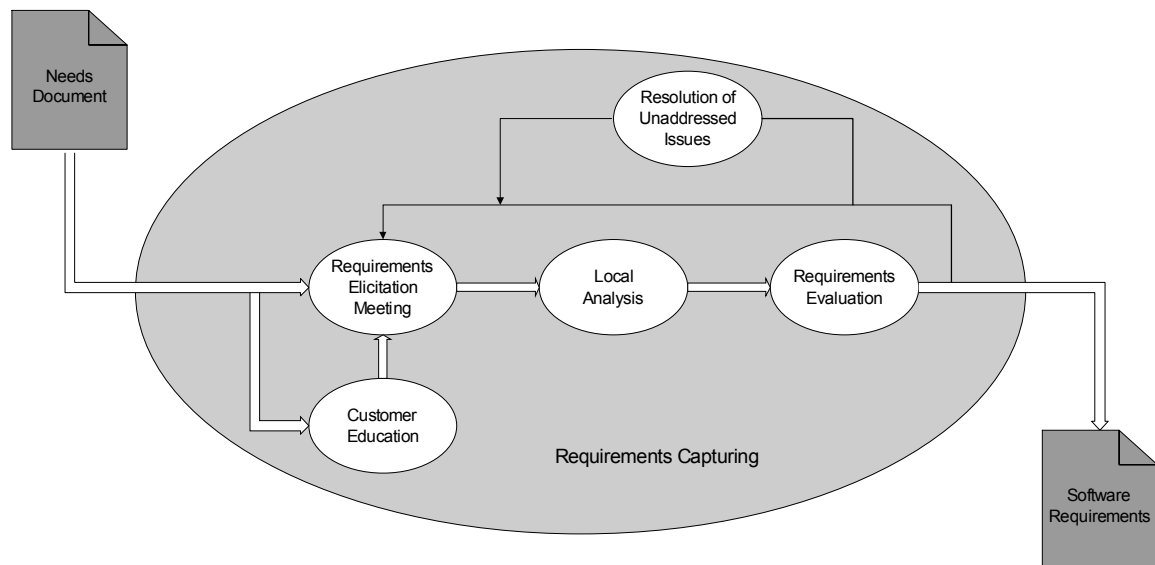


Figure 3.11: Requirements Capturing

3.3.1 Customer Education

Educating the customer is an optional phase, and as indicated in Figure 3.11, it can be *by-passed*. However, we do recommend that all participants involved in requirements generation, including users and customers, be familiarized with the requirements generation process. Familiarity with the model allows the participants

to understand the flow of the model, and learn what is expected of them. Specifically, we recommend that the participants be *educated* about the following aspects of the requirements generation process:

- A brief introduction about Requirements Engineering, the importance of requirements, and what does and does not represent a requirement
- An overview of the Synergistic Requirements Generation Model, describing the participatory approach and roles in the overall development process
- The importance of cooperation in information gathering activities conducted at the start of the process
- The preparation involved when participants are requested to attend elicitation meetings.

We understand the time and cost constraint that this phase may impose on the project, as well as the possibility that the customer may not want to be directly involved in the process. Therefore, this phase is not a mandatory activity.

3.3.2 Requirements Elicitation Meeting

Once the problem has been captured and the needs have been elicited, the focus shifts on eliciting the solution – “*What functionality is the solution going to offer/possess?*” Requirements elicitation meetings are conducted to elicit information from stakeholders. The primary objective is to identify and capture requirements as communicated by the customer and the users. We present the Structured Meeting Approach in the section below with the intent to structure interactions between stakeholders.

3.3.2.1 The Structured Meeting Approach

The “Structured Meeting Approach” is a critical part of the Synergistic Requirements Generation Model. It provides the means and the guidance for realizing the *participatory approach* advocated throughout this thesis. We strongly recommend that this approach be followed whenever any interaction between the stakeholder and the requirements engineer is planned.

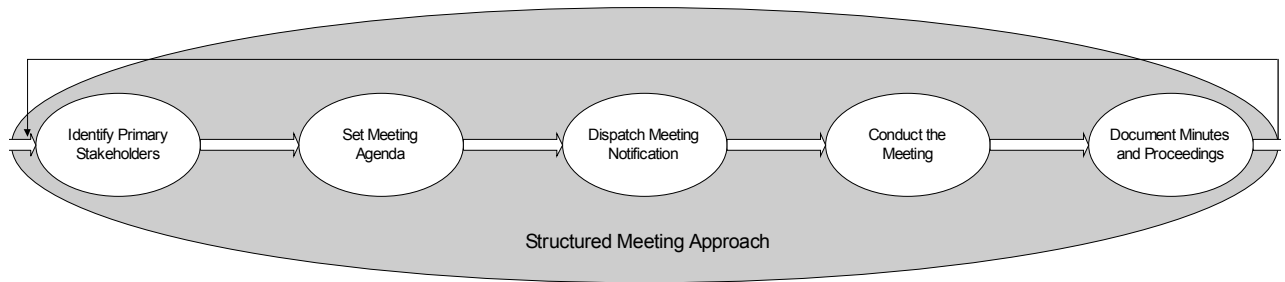


Figure 3.12: The Structured Meeting Approach

Figure 3.12 provides a snapshot of the activities involved in the Structured Meeting Approach. The first activity is the identification of the meeting participants. Identification can be done as per the JAD or the PD methodology (Section 2.2.1.3.1: JAD and Section 2.2.1.3.2: PD). As mentioned previously, JAD and PD differ in the selection of participants. To briefly reiterate, JAD advocates participant representation from an entire *cross-section* of the hierarchy of the organization, while PD advocates selection of participants from the same hierarchical level. The choice of the participant selection method is left to the requirements engineer. The selection is supported by the *Hierarchy Generator* – a tool devised as a part of the suite of tools developed for supporting *itSRGM*. A complete discussion of this tool is provided in Appendix A.

Once identified, the participants are sent a meeting notification outlining the agenda of the forthcoming meeting. Definitive tasks may be assigned to all or particular participants in context to the preparation for the meeting. For example, some stakeholders may be assigned to present information regarding a viable approach or an idea to the other participants. Pre-defined templates (Appendix A) can be used to prepare meeting agendas. Once prepared, the meeting notification is dispatched to all stakeholders, as per the selection made earlier in the process.

The meeting is conducted as per the guidelines imposed by the JAD, PD, or other approaches, depending on which is more appropriate. It is imperative that the proceedings of the meeting be documented. These proceedings are an important source of requirements, albeit an unstructured one. After the meeting, these proceedings must be structured and circulated amongst all meeting

participants and relevant stakeholders, with the intent to receive feedback about the correctness of information represented.

3.3.3 *Local Analysis*

The Local Analysis process represents an innovative approach to the analysis of software requirements. The term ‘Local’ implies that activities defined within this phase are conducted as and when requirements are generated. Local analysis aid a ‘global’ analysis process. Information associated with every requirement derived during “local analysis”, is collectively analyzed in the Requirements Analysis process (Section 3.4).

Figure 3.11 indicates an iterative process for requirements generation. For a reasonably large system, it is almost impossible to elicit requirements for the entire software system in one sitting. Multiple meetings, involving a diverse set of stakeholders, must be conducted to ensure complete coverage. During these meetings, requirements are generated and documented. Local analysis associates attributes, as indicated by the activities in Figure 3.13 below, to the appropriate requirement. The requirements engineer may select certain attributes while ignoring the others. The process structure for the Local Analysis activity is adapted from Alan Davis’s Requirements Triage [Davis 1999]. The more information associated with each requirement, the more informed all stakeholders will be with respect to making decisions about the overall project.

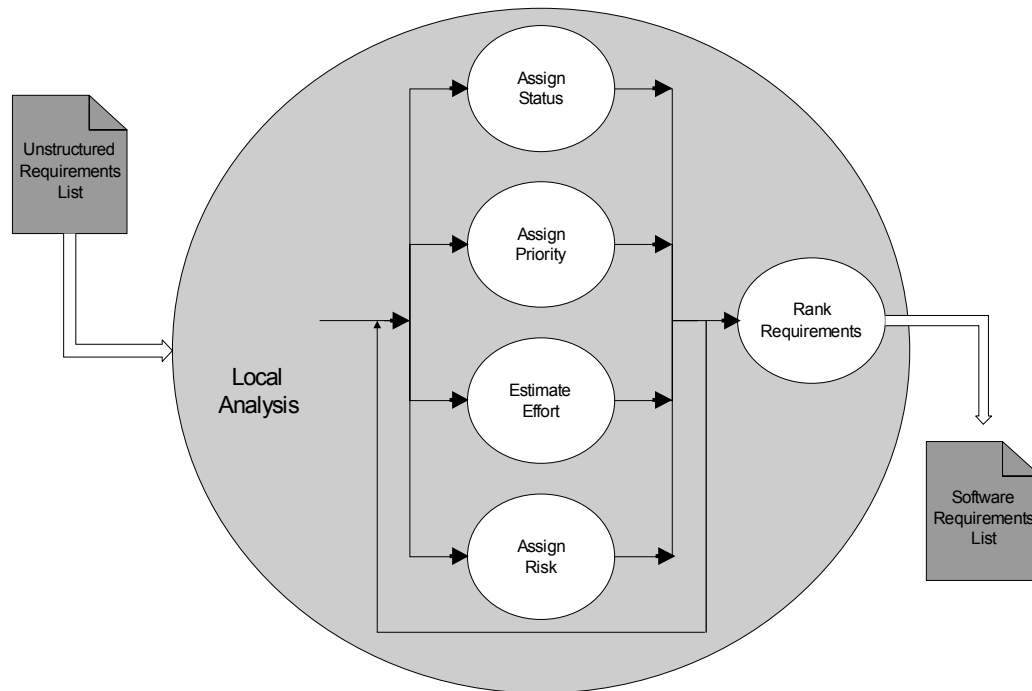


Figure 3.13: Local Analysis

The Synergistic Requirements Generation Model advocates *inherent* analysis activities. Appendix A indicates useful artifacts and guidelines to associate attributes with requirements.

3.3.4 Requirements Evaluation

Requirements Evaluation is the final sub-phase within the Requirements Capturing process. The Evaluation phase is entered immediately after ‘local’ analysis is conducted on the requirements generated in the preceding requirements elicitation meeting.

Requirements Evaluation is primarily conducted to verify the adherence of requirements to quality attributes. This helps uncover inconsistencies and redundancies. Quality attributes such as those listed in Figure 3.14 are tested. This may be done by the requirements engineer, using his/her own experience, or in consultation with the stakeholders and QA experts.

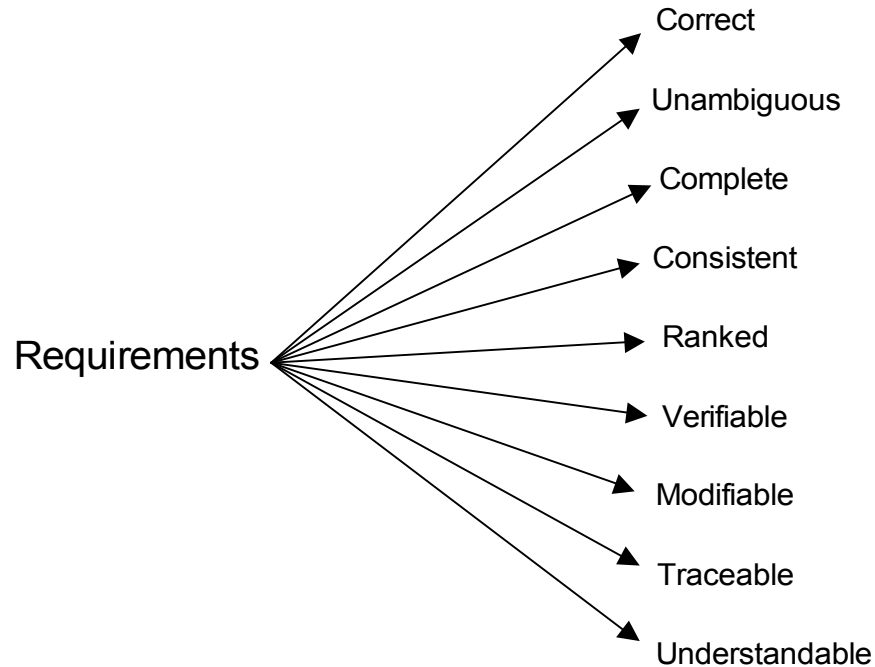


Figure 3.14: Requirements Quality Attributes

The Requirements Evaluation sub-phase also determines the need for an additional iteration through the entire Requirements Capturing process, or whether a transition to the Requirements Analysis phase can commence. At the completion of the requirements evaluation sub-phase, the requirements engineer, in consultation with the customer must determine if the requirements capturing process is complete. If the answer to this question is “no”, at least one more iteration through the requirements capturing phase is necessary. On the other hand, if all stakeholders involved feel that all requirements have been captured meeting the *correct* intent, a transition may be made to the Requirements Analysis phase. The following exit criteria may be helpful to the requirements engineer to determine the completion of requirements capturing (adapted from [Groener 2002]):

- Checking requirements’ attributes
- Finding agreement among all stakeholders about all requirements
- Ensuring no open or unresolved issues remain

3.4 Requirements Analysis

The Requirements Analysis phase incorporates processes that ensure the correct software product is being built. A ‘correct’ software product has the “correct mix of price, cost, and features.” Based on the requirements generated from the Requirements Capturing process, certain ‘global’ activities are performed that evaluate the ‘fate’ of the software product, from various standpoints. A development effort not incorporating this phase will be subject to considerable risk, perhaps even resulting in the loss of customer interest.

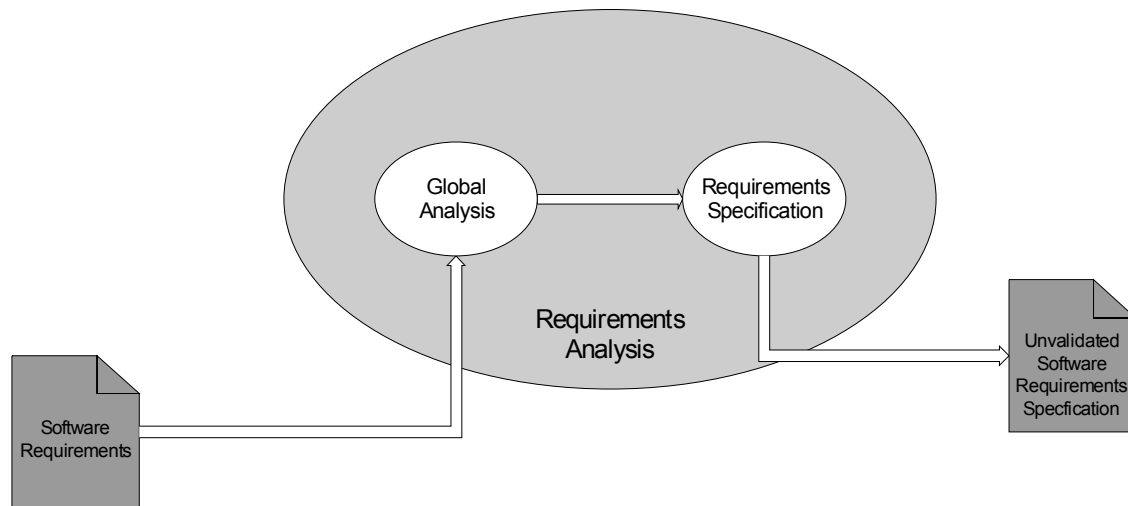


Figure 3.15: Requirements Analysis

As per the Synergistic Requirements Generation Model, Requirements Analysis is comprised of two sub-processes – Global Analysis and Requirements Specification. Global Analysis assesses the viability of the requirements from multiple perspectives. The Requirements Specification process offers guidelines to document the requirements. Each of these processes is discussed in greater detail.

3.4.1 Global Analysis

The activities performed in the Global Analysis phase are similar to those conducted during Local Analysis (Section 3.3.3). They differ, however, in scope. Local Analysis is performed on requirements as and when they are generated just after the requirements elicitation meetings, while Global Analysis activities are conducted on the complete set of requirements.

The Global Analysis set of activities also represents a *suite of decisions* that *must* be taken, when requirements/features are generated. These decisions are listed after the analysis of various attributes.

The information (and attributes) tabulated during Local Analysis are *collectively* analyzed to assess the overall progress of the project and whether the project makes “sense” from the development, market, sales and financial perspectives. Figure 3.16 gives an indication of the iterative approach used to conduct the Global Analysis activities. A brief description, adapted from [Davis 1999], of the activities performed follows.

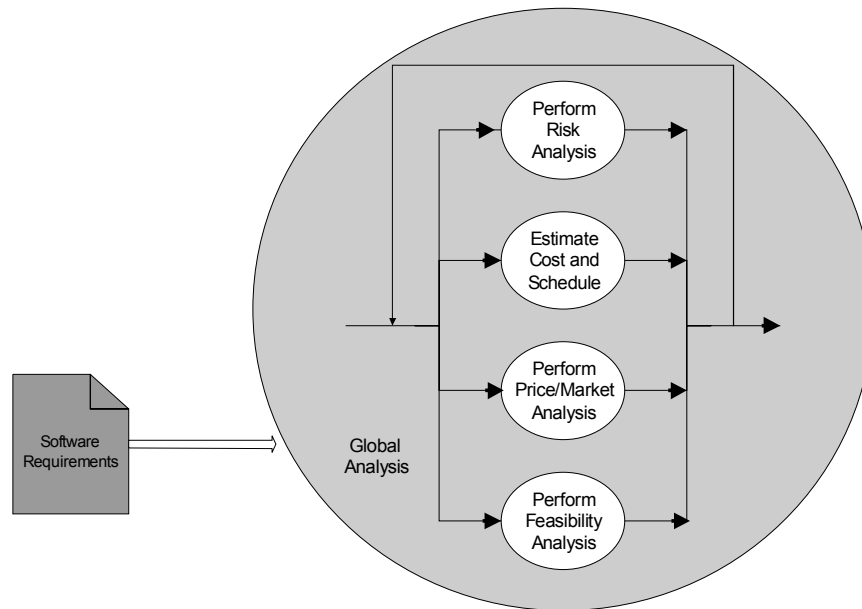


Figure 3.16: Global Analysis

Perform Risk Analysis: The Local Analysis Activities assigned a risk factor in the form of a rank to the risk associated with each requirement. This is now weighed in collectively with all the requirements generated so far.

Estimate Cost and Schedule: Determine the effort and time required to satisfy potential features. This is done in consultation with the developers.

Perform Price/Market Analysis: Determine optimal price to charge to customers, or market demand (identify customer types, buying characteristics, marketing, etc.), in case of shrink-wrapped product

Perform Feasibility Analysis: Determine the feasibility and the profitability associated with the software product. Based on this analysis, a 'go-no go' decision is to be taken

In performing Global Analysis, three distinct perspectives must be considered:

- Development
- Marketing and Sales
- Financial.

Representative questions germane to each perspective include:

Development Perspective

- Can the selected features be built?
- By when can the selected features be built?
- How many resources are required to build the selected features?
- What kinds of resources are required to build the selected features?
- How much risk can the organization tolerate?
- What are the ramifications of being late or being over-budget?
- Are any requirements conflicting with each other?

Marketing and Sales Perspective

(Note: This perspective will be included only when the software development effort deals with the development of a shrink-wrapped product)

- How big is the market potential for a product with the selected features?
- At what price can we sell a product with the selected features?
- When does the customer want the product?
- How does the proposed product compare with the competition?

Financial Perspective

- How will the use of the product save (or earn) the company each time it is used?

- How much is the customer willing to pay for the software development effort?

Capturing and documenting these perspectives is critical to the progress of the development effort considering the competitiveness in the software industry.

3.4.2 Requirements Specification

Once requirements have been captured and analyzed they must be organized and documented in the form of a document, referred to as a “*Software Requirements Specification*” (SRS). Doing this is especially critical since the written documents serve as a contractual agreement between the customers and the developers.

The SRS serves the following purposes:

- It precisely states the functions and capabilities that a software system must provide and the constraints it must respect
- It provides a basis for all subsequent project planning, design and coding, as well as the foundation for system testing and user documentation
- It describes the intended user interface of the system
- The SRS should NOT contain design, construction or testing details other than known design and implementation constraints.

It is important that the SRS be comprehensive – *all* requirements should be included. The requirements engineer and the customers should make no assumptions. If any desired functional or non-functional requirement(s) is omitted from the SRS, then it is not considered as a part of the contract and no one should expect it to appear in the system, once implemented [Weigers 2001].

Requirements that have been gathered must be documented in adherence to a standardized format that makes information retrieval easier for designers, developers and customers. Formats have been recommended by standards organizations such as the IEEE (Appendix C). The Synergistic Model for Requirements Generation does not recommend any specific template, as long as the format is agreed upon by all stakeholders, is unambiguous and captures all information required for future development activities.

3.5 Requirements Validation

The Synergistic Requirements Generation Model offers ‘interwoven’ verification and validation support to the requirements generation process. A discussion is provided in Section 2.4.1. Initially, the investigation conducted in the Problem Synthesis phase verifies whether the correct problem is being addressed. Next, the needs generated from the problem elements are verified against the problem statement. These needs are then used to elicit software requirements. Software requirements are continually evaluated, as and when they are generated, ensuring that they adhere to quality standards and meet the customer intent. Finally, these requirements are collectively integrated to form the Software Requirements Specification (SRS). The requirements contained in the SRS are then validated against the needs and problem statement.

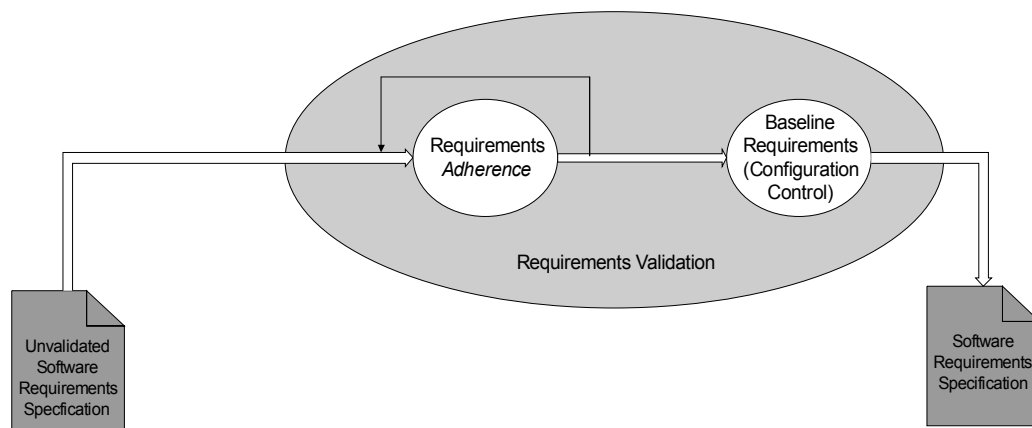


Figure 3.17: Requirements Validation

The Requirements Validation phase comprises activities for validating the SRS. It includes activities intended to ensure that:

- the SRS correctly describes the intended system behaviors and characteristics
- the software requirements were correctly derived from the system requirements or other origins
- the requirements are complete and of high quality
- all views of the requirements are consistent

- the requirements provide an adequate basis to proceed with product design, construction and testing, with a high degree of confidence.

As indicated in Figure 3.17, Requirements Validation comprises of two sub-processes – Requirements Adherence and “Baseline” Requirements. Each of these has been described in more detail in the sections that follow.

3.5.1 Requirements Adherence

Requirements Adherence (Figure 3.18) comprises of activities that ensure the adherence of requirements to expected conditions and standards. They include activities such as Requirements Traceability and Formal Technical Reviews to seek mutual agreement about the overall SRS amongst customers and developers. Finally, in the event of an *anomaly* certain procedures must be followed to ensure that requirements are reworked and edited properly.

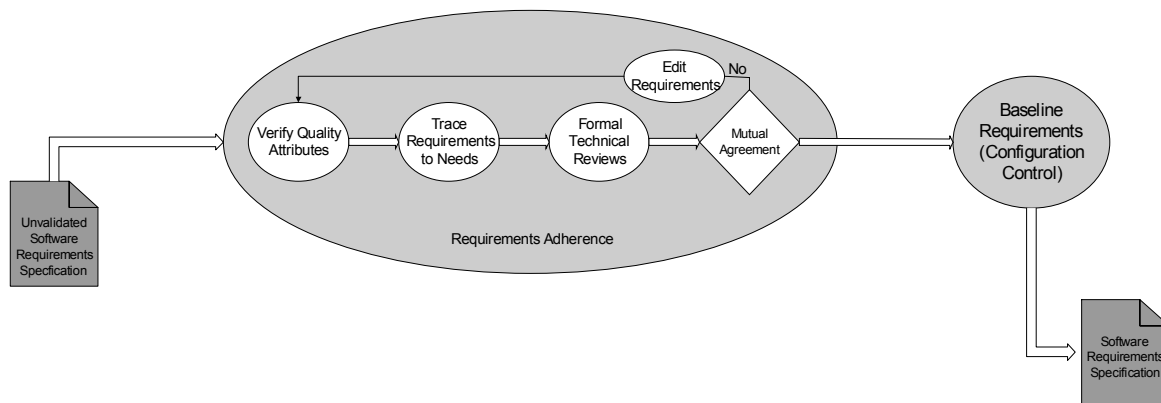


Figure 3.18: Requirements *Adherence*

3.5.1.1 Verify Quality Attributes

The importance of the requirements specification with respect to the overall software development lifecycle cannot be over-emphasized enough. It is hence essential to verify all quality attributes of the overall document. We recommend that ‘independent’ quality analysts be involved to conduct this activity. The

requirements engineer in consultation with experts may also perform the requisite activities.

Non-functional Requirements: A common mistake committed by customers and developers is that they focus on specifying functional or behavioral requirements, often not specifying explicitly, their expectations about *how well* the software product will work. These characteristics include how easy the product is to use, how quickly it executes, how reliable it is, and how well it behaves when unexpected scenarios arise [Weigers 2001]. Such characteristics, known as quality attributes, are referred to as the system's non-functional (or non-behavioral) requirements, and must be specified, by the requirements engineer, in consultation with the customers and developers. Robert Charette [Charette 1990] points out that, "In real systems, meeting the nonfunctional requirements often is more important than meeting the functional requirements in the determination of a system's perceived success or failure." Often, customers do not volunteer their nonfunctional expectations. However, it is important that this information be solicited and be made a part of the SRS. Non Functional Requirements have a strong bearing on the overall quality of the system, as they dictate the expectations from the implementation of the system.

Section 3.3.4 and Figure 3.14 enumerates quality characteristics that requirements must possess. For any product, it is not possible to incorporate all quality attributes. The requirements engineer must prioritize and quantify the quality attributes, envisioned by the customer. At some point in time, the stakeholders should be asked to prioritize the quality attributes, starting with the ones they deem most important. A mean of the ranks assigned by stakeholders can be used to determine the relative importance of each attribute. The SRS can then be *molded* to incorporate the particular quality attribute(s). For example, for an interactive application, engaging heavy user inquiries (e.g. a self-checkout kiosk at a grocery store), usability may be of paramount importance. In this case, certain *quality requirements* may be included in the SRS that account for this aspect. On the other hand, for an embedded software system, performance may of primary concern.

3.5.1.2 Requirements Traceability

Requirements Traceability entails the representation of links among requirements, needs, and other system elements. Requirements Traceability is an important component of Requirements Verification, as it accounts for the fulfillment of all the needs established earlier for the system. Requirements Traceability Matrix links functional requirements to needs, and needs to problem elements. In this way, all the problem elements and needs can be guaranteed to be associated with one or multiple requirements.

Requirements Management tools, such as those described in Appendix B, simply support the traceability process. We recommend the use of one of these tools for specifying and creating and manipulating traceability links amongst all system elements.

3.5.1.3 Formal Technical Reviews

A Formal Technical Review (FTR) is a ‘team effort’ to eliminate problems in the deliverables through group reviews. Formal Technical Reviews include activities that focus on overall examination of the requirements specification. They provide a way for stakeholders to agree on the requirements.

There are multiple ways of conducting technical reviews. Informal approaches include the distribution of the requirements specification to several peers and stakeholders to look over and provide feedback.

Whereas informal reviews are performed in an *ad hoc* fashion, a Formal Technical Review (FTR) follows a well-defined process with a prescribed sequence of steps. A FTR should result in a written report that identifies potential defects and anomalies. A brief overview of FTRs, as adopted in the Synergistic Requirements Generation Model follows.

Reasons to conduct Formal Technical Reviews:

- To ensure that the requirements specification is complete and unambiguous
- To ensure that the requirements meet the customer intent

- To detect and correct errors and discrepancies early in the development process.

One or more of the following approaches may be used to conduct FTRs:

- Peer Reviews
- Inspections (by upper management or independent agencies)
- Walkthroughs
- System Prototype Presentation.

The method of conducting an FTR Meeting is similar to that of the Structured Meeting (Section 3.2.3.2.1). The objectives however differ:

- Defect Removal in the requirements specification
- Assess quality of the requirements specification and the process.

3.5.1.4 Reworking (Editing) Requirements

Formal Technical Reviews should provide a written report identifying irregularities in the SRS. These must be corrected immediately, as uncorrected requirements will be expensive to fix down the road. Once changed, the associated stakeholders must follow up on the rework. As indicated in Figure 3.18, an iterative process follows, in which the edited requirements are again subjected to the rigorous quality checks to ensure that the changed requirement(s) has not introduced any conflicts and contradictions with those previously defined.

3.5.2 Baseline Requirements (Configuration Control)

Changes in requirements are inevitable. These changes may occur during the design, coding or testing phase. No requirements generation model can claim to eliminate these changes completely. However, it is our conjecture that employing the Synergistic Requirements Generation Model will significantly reduce the amount of rework and editions required subsequently in the software development lifecycle.

Once all changes are approved and incorporated, the final step of the Synergistic Requirements Generation Model is to baseline the SRS, as per the organization's *configuration control policy*. All stakeholders, including the senior management,

must agree upon this configuration control policy. The following guidelines can be employed while establishing the configuration control policy:

- Ensure that all requirements are correctly numbered and related requirements are flagged
- For initiating a change, a formal Change Request form must be filled in and circulated by the stakeholder (including developers and testers) to all other stakeholders, outlining the change, and providing a detailed rationale for incorporating the change
- Depending on the extent of change requested, a meeting may be convened to discuss the change and formally approve it via consensus. On the other hand, for a minor modification, the change may be incorporated and communicated to all stakeholders
- The newer version of the SRS must be circulated amongst all associated development teams and stakeholders. The first page must specify the changes made to the overall SRS.

In this chapter, we have provided detailed descriptions of the processes and activities of the Synergistic Requirements Generation Model. Throughout the chapter the focus was to emphasize the *holistic* approach – coverage for the entire requirements generation process. The model started with the Conceptual Overview process, producing a Problem Description Document. This was followed by Problem Synthesis, during which the *real* problem was identified, analyzed and decomposed. Needs were subsequently elicited for the problem. Requirements were elicited for these needs in the Requirements Capturing process. Several guidelines and approaches were outlined, all attempting to structure the interaction taking place among stakeholders. In-depth analysis was then conducted on the elicited requirements, examining them from several perspectives. Finally, the requirements were validated, ensuring that they met the customer intent.

We do recognize the limitations in the applicability of the SRGM. The SRGM was conceived as a part of the “Requirements” phase in the Waterfall Model [Royce 1970]. The intent of this conception was to focus, characterize and refine the activities performed within the Requirements Generation phase, with minimal concern about other

phases of the Software Development Lifecycle. However, we do recognize the limitation of the SRGM, in its current form, of being applied to other development paradigms. We do conjecture that the SRGM can be modified to ‘fit’ the evolutionary paradigm.

Chapter 4 introduces the second component of the Synergistic Approach to Requirements Generation – the interactive tool. The tool essentially is a codification of the Synergistic Requirements Generation Model.

Chapter 4 An Interactive Tool for Modeling The SRGM (*itSRGM*)

4. SOFTWARE REQUIREMENTS GENERATION TOOL

The Synergistic Requirements Generation Model captures requirements generation processes and activities. It commences with the collection of the initial input(s), followed by identifying and documenting the problem, eliciting and specifying the requirements and finally, in-depth analysis and verification, resulting in the culmination of the Software Requirements Specification (SRS).

The second component of our holistic approach is a tool – an interactive application codifying the model. The tool, christened “An Interactive Tool for Modeling the SRGM” is an innovative application intended to provide the requirements engineer with tools, artifacts and guidelines supporting the requirements generation process embraced by the Synergistic Requirements Generation Model. It also provides a visual representation of the activities being performed, and is capable of presenting those activities at varying levels of detail.

This chapter describes the design decisions and components of *itSRGM*. The components are a reflection of those identified in the Synergistic Requirements Generation Model (Chapter 3). It is the intent of this chapter to describe the tool and not reiterate a description of the processes. Appendix A provides tabulated synopsis of all major processes.

4.1 Need for An Integrated, Interactive Tool

Current approaches to managing requirements provide support to the process, but in a “disconnected” fashion. There are tools⁸ that help the requirements engineer document and “specify” the requirements. Other tools allow analysis, and “what-if” scenarios to be performed on requirements, enabling stakeholders to gain varied perspectives of requirements and the implications of their incorporations. However, these tools only ‘*come into action*’ once requirements are already specified, thus pulling attention away from the more *important* elicitation and investigation activities.

⁸ Six popular Requirements Management Tools are surveyed in Appendix B

Moreover, there is no *one* requirements management tool or process that accounts for all the phases of the requirements generation process. A tool can only claim to provide complete support to the requirements engineering process, if it offers end-to-end support. We have conceived a tool that provides support to the requirements engineer, starting with the initial input culminating in the production of the SRS. This integrated approach ensures that all components required to generate requirements are provided by one single, integrated tool, while also providing the flexibility in selecting among possible requirements generation tools and approaches (Chapter 2) as demanded by the project characteristics.

We now provide a brief introduction to the concept of “Environments,” which we envision will be the ultimate embodiment of the Interactive Tool for Modeling the SRGM.

Introduction to Environments

Quintessentially, a requirements generation environment provides the requirements engineer with the *appropriate* mechanisms to assist in requirements analysis, documentation and verification activities, while providing guidelines for elicitation. An environment also provides the means to implement a process and enforce procedures guiding the requirements engineer. The requirements engineer is also provided a suite of software tools needed to gather and analyze information. Such tools support activities *defined* by the process and *enforced* by the environment. An explicit mention must also be made of the flexibility that is required to be demonstrated by the environment. Various approaches and tools supporting the requirements process are available. An environment must be flexible enough to accommodate various approaches or tools, depending on the stakeholders, or specific project needs.

In the next sub-section, we enumerate certain “meta-requirements,” or features that were considered when conceiving *itSRGM*. The section also provides detailed information about the design and the implementation of *itSRGM*.

4.2 Developing the Interactive Tool For Modeling The SRGM

This section outlines the development of *itSRGM*, starting with the high level requirements, specifying the design elements and the development platform.

itSRGM Meta-Requirements:

“Meta-requirements” list requirements for a requirements generation tool. Motivated by the needs explained before and in the previous chapters, the following attributes were considered and incorporated into *itSRGM*.

- The requirements generation tool shall guide the user (requirements engineer) along predefined processes and/or activities supporting requirements generation. This predefined path is reflected in the Synergistic Requirements Generation Model (Chapter 3)
- The requirements generation framework should have the flexibility to accommodate alternatives (Chapter 2) to appeal to a wide spectrum of users and projects. However, the structure of the underlying framework should be enforced, i.e. the *type* of activities and processes must be specified and followed as per the Synergistic Requirements Generation Model. In other words, the framework, on which the tool is based, must be independent of any specific alternative

For example, in the structured meeting approach, participant selection may be facilitated as per the JAD (Section 2.2.1.3.1) or the PD (Section 2.2.1.3.2) methodology, but it is imperative that participants be selected in an organized manner before the meeting

- The interface for the tool must be intuitive. Activities performed should be accompanied with a visual feedback, confirming the action.
- The platform shall provide a set of tools, artifacts, guidelines, templates and checklists that aid the processes defined within the framework
- In the spirit of the *participatory approach* discussed throughout this thesis, the tool must facilitate communication among stakeholders. Tools and guidelines should be provided to aid that communication.

itSRGM Design Components:

While conceiving *itSRGM*, we recognized that variegated information had to be organized and presented in a manner that would convey a synergistic approach, that is, all processes depicted in the Synergistic Requirements Generation Model (Chapter 3) had to be seamlessly integrated, with a common look-and-feel supporting the intended functionality. To achieve this, the following constituent design elements have been incorporated in *itSRGM*:

- Event Sequencing, which provides the capability to link events together in a serial fashion
- Process Decomposition, which allow the user to delve into greater details of a specific process
- Abstraction, which allows the user to view the context within which the low-level activities are performed
- Decision Events, which provide the capability to incorporate *decision points* into the processes
- Event Repetition (Iteration), which enables the definition of an interactive scenario when one or more events are to be repeated multiple times
- Instruction Components, which are represented visually and which suggest activities that guide the user through the requirements generation process
- Artifacts, which are employed by the user performing the associated activities, e.g. preparing the dispatching the meeting agenda/notification
- Hyperlinks and Mouse-over effects, which provide the “glue” to link the above design components together to form the integrated tool topology comprising of component sequences, decision components and event repetitions.

These design elements and components will be further explained in section 4.3.

itSRGM Implementation:

Implementing *itSRGM* was a challenge, as one of the most important considerations was that of integration. Instead of ‘reinventing the wheel’ existing tools were integrated

into *itSRGM*. The issue was to invoke these tools⁹ at the appropriated instant of time. The following implementation features were addressed while developing *itSRGM*:

- An interactive, web-based platform facilitating sharing of information about the process, as well the products and deliverables (requirements, needs, etc.)
- Visual representation of the processes and activities currently being executed, as well as those completed and forthcoming
- Two window frames outlining the progression – the top frame providing visual feedback about the process(es), and the bottom frame providing navigation, guidelines and interaction
- Embedded hyperlinks performing either one of the following activities, depending on the place they are invoked:
 - Decomposition of the process
 - Invocation of the relevant tools/artifact aiding the (low-level) process
 - Progressing through a process
 - Obtaining information about processes
 - Uploading artifacts and deliverables

(These aspects are explained in greater detail in section 4.3)

- Automatic invocation of tools and artifacts depending on the process being currently addressed
- Mouse-over effect for every ‘process bubble’ providing a short statement of purpose and intent of the process
- A complete (Level 1) snapshot of the requirements generation process, as conveyed by the Synergistic Requirements Generation Model (Chapter 3). This provides the stakeholders with the context of operation and allows them to focus in on any process, or abstract back to a higher level
- A Preview mode providing an interactive overview of the entire requirements generation process
- Detailed descriptions accompanying every ‘process bubble.’

⁹ Tools have been discussed in Appendix B

The next section provides a more detailed look at the individual components of the process implemented in *itSRGM*. Information about the individual processes is provided in Chapter 3 and Appendix A. The intent of this section is to explain the flow of specific processes, accompanied with screenshots of *itSRGM*.

4.3 Components of the Requirements Generation Tool

Chapter 3 described the *what* aspects of the synergistic approach to requirements generation, i.e. a description of the processes that are a part of the approach. This section takes a look at the components, and *how* the tool is implemented. Screenshots are embedded in the description of the processes, to aid understanding. An initial prototype of *itSRGM* can be found at <http://csgrad.cs.vt.edu/~rsud/Environment>

Components contained in *itSRGM* can be grouped into five categories: (1) Processes, (2) Visual Notations, (3) Mouse over effects, (4) Detailed Process Description, and (5) Decision Points and Iterations.

4.3.1 Processes

From the structural representation perspective, *itSRGM* incorporates three types of processes. The distinction is made on the basis of whether the process is associated with any input, output or is a standalone process, *i.e.* without any inputs or outputs.

- a. Input Process – If a specific process expects an input document, the user is explicitly asked to identify the expected document. Only when the document location is specified is the user allowed to continue in the process
- b. Output Process – If a specific process generates an output, *e.g.*, in form of a document, artifact or a deliverable, the user is assisted in creating that document or is asked to specify its location. Progression is prohibited until the document is created/identified
- c. Standalone Process – If a process neither requires nor generates an artifact, the user is allowed to proceed directly to the next activity, *assuming* that the activities advocated by the process are performed.

All process descriptions adhere to the following template:

<Process Name>
Objective: <a brief objective of the process>
Guidelines: 1. <Guideline 1> 2. <Guideline 2> 3. <Guideline 3>
<Instructions to progress through to the next process>

Figure 4.1 provides an example of the Process Description, in the form of a partial screenshot of *it*SRGM. The process indicated is “Initial Customer Interaction,” which is conducted typically at the start of the requirements generation process.

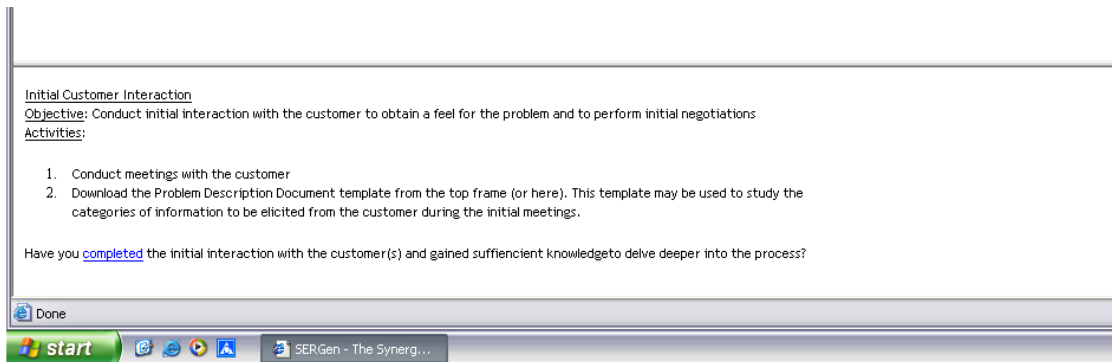


Figure 4.1: Partial Screenshot of *it*SRGM, indicating the representation format of a typical process description

Figure 4.2 provides a partial screenshot of the “submit pane”, in which the user is asked to specify the location of a document. The current prototype accepts any input, but future implementations may incorporate elaborate error checking, in the form of ‘text parses,’ to ensure that the document specified is the one expected.

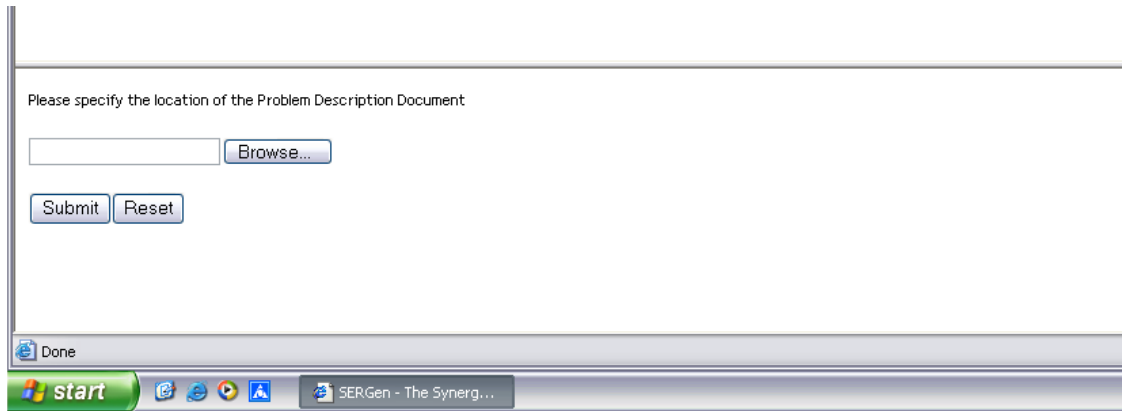


Figure 4.2: Partial Screenshot of *itSRGM*, indicating the “submit” pane

(Note that all the information represented in the template, and the “submit” pane is always displayed in the bottom frame of the page. The bottom frame of the tool is also the primary way of progressing through the process)

4.3.2 Visual Process Notations

Corresponding to the Process Description representation in the bottom frame, the top frame displays the corresponding flow visually, in the form of “process bubbles.” The process bubble which is currently *active* (the activities for that process are being performed) is highlighted in red, while the others are “grayed” out. This representation is intuitive since it displays the current process being performed and also provides the user the context of operation.

Figure 4.3 provides a screenshot of the decomposition of Problem Identification, within the Problem Analysis process. Notice that the process bubble highlighted in the top frame corresponds to the activities, being performed in the bottom frame.

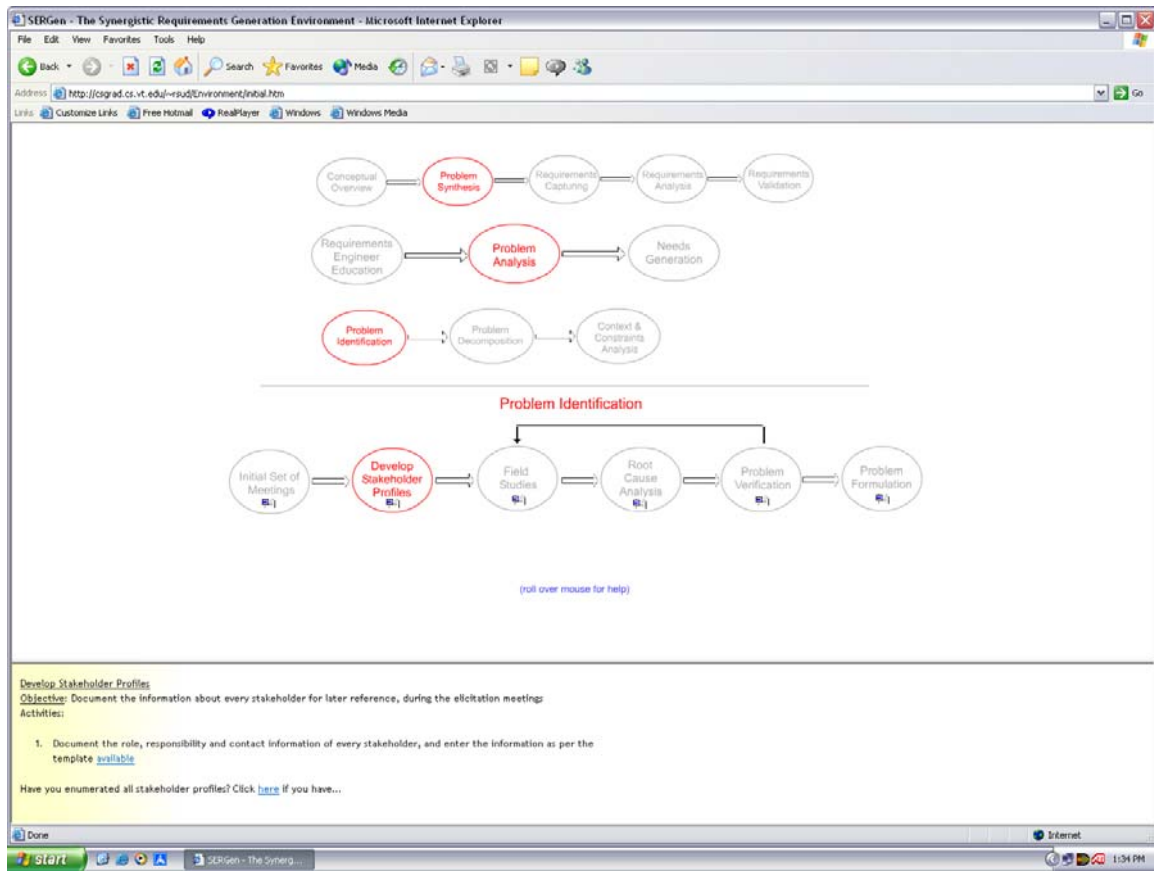


Figure 4.3: Screenshot representing “Develop Stakeholder Profiles”, of the Problem Identification process. Note the overall process represented on the top of the page, providing the overall context. Also notice the “graying” of inactive bubbles, and the red highlighting of the “active” bubbles

4.3.3 Mouse Over Effects

The ‘Mouse over’ effect provides additional information about each ‘process bubble’ displayed in the top frame. The user simply has to move the cursor into the process bubble, and a short description of that process will fade in. The description will fade out once the cursor is rolled outside the bubble.

Figure 4.4 provides a partial screenshot of the mouse-over effect. The reader is advised to visit <http://csgrad.cs.vt.edu/~rsud/Environment> to view the actual implementation of this feature.

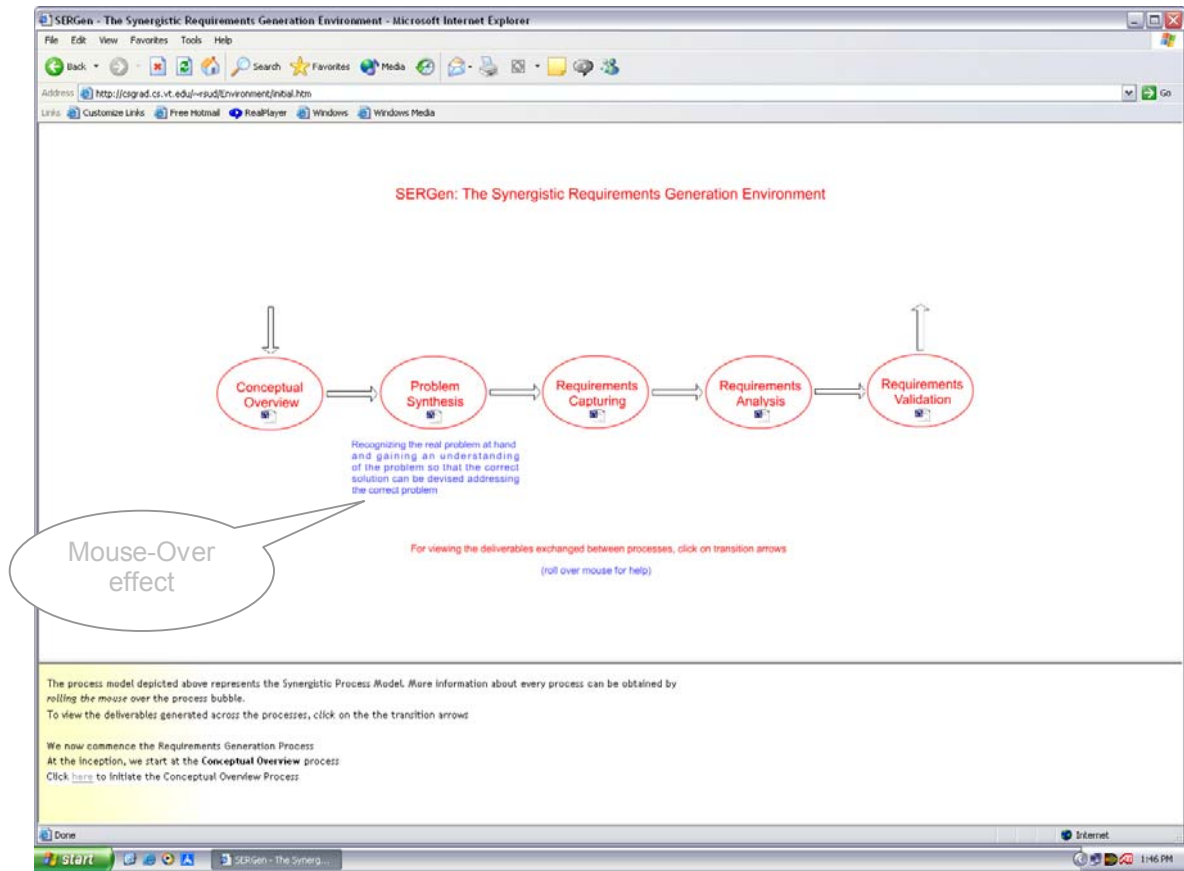


Figure 4.4: Screenshot of *itSRGM* indicating the 'mouse-over' effect for the Problem Synthesis process bubble. The specific screen is presented to the user at the start of the process

4.3.4 Detailed Process Descriptions

itSRGM provides detailed process descriptions, through the visual representations in the top frame of the application. The  icon associated with every 'process bubble' represents the detailed representation associated with that process. Simply clicking on the icon enables the download of a document providing the detailed descriptions. A shorter version of the descriptions is provided in the bottom frame (as indicated in Section 4.3.1) and is displayed as the user progresses through the process. The mouse-over for each 'process bubble' also provides a brief synopsis.

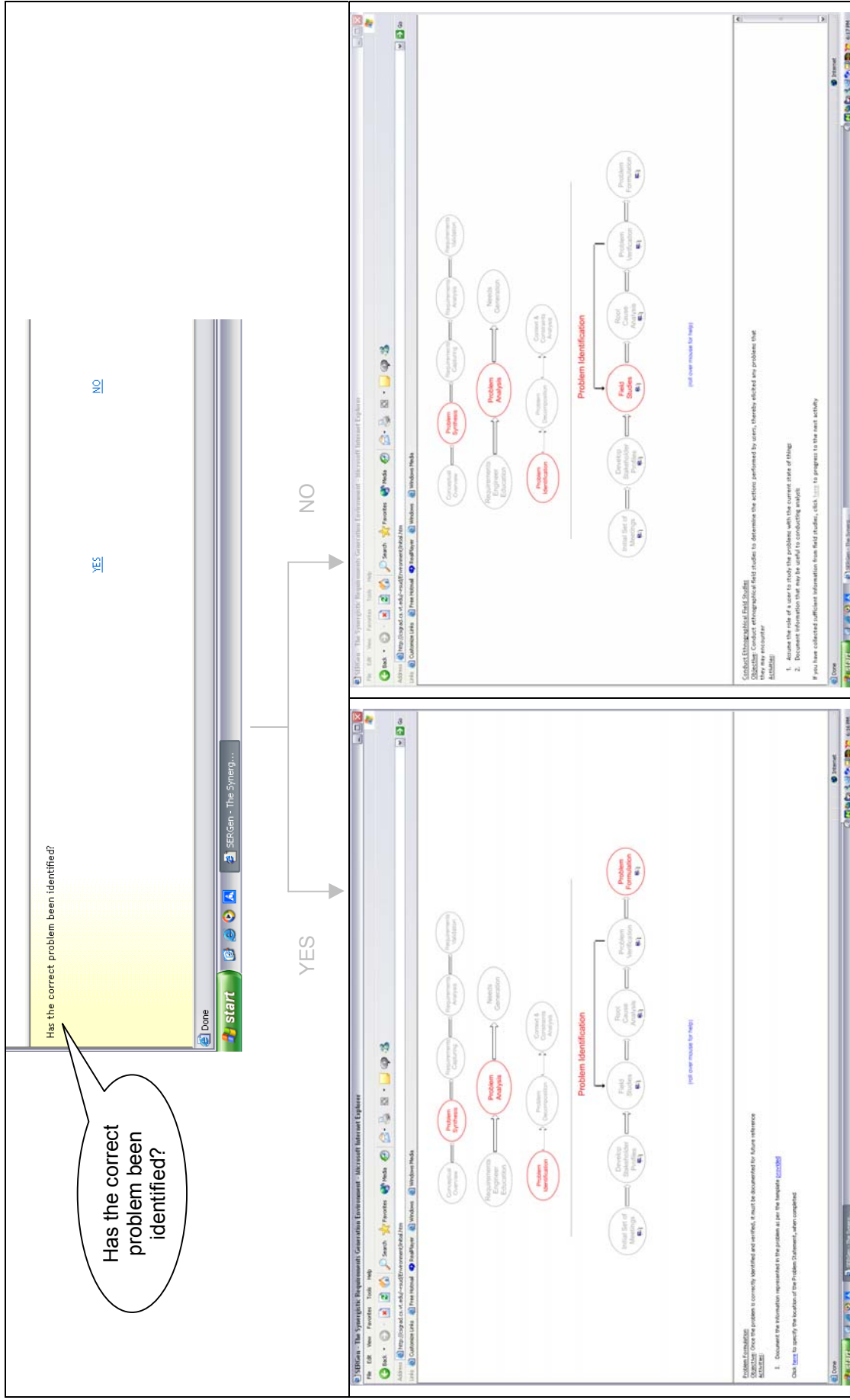


Figure 4.5: Decision Point for verifying the correctness of the problem

4.3.5 *Decision Points and Iterations*

The Synergistic Requirements Generation Model (Chapter 3) incorporates certain ‘iterative’ activities – activities which are performed multiple times to either elicit more information, or to refine the existing information. These iterative capabilities are also captured within *it*SRGM.

Figure 4.5 displays a series of screenshots of the iterative activity, enabling a user to verify whether the correct problem has been identified. Simply enough, the user is led to a ‘decision point’ posing a question whether iteration is required. In case of an affirmative reply, the user is led back to the activity at the beginning of the iteration, as advocated by the model. In case the activities performed previously do not warrant iteration, the user simply negates the decision point and is subsequently led to the next activity in the process. Once the location of the expected document is specified, the user performs the other activities routinely.

Completion of activities, which do not lead to a decision point, is sought by posing questions in the bottom frame. The user is expected to progress to the next activity only after the activities of the preceding process are complete. Figure 4.6, below, provides a partial screenshot indicating a “meaningful” question.

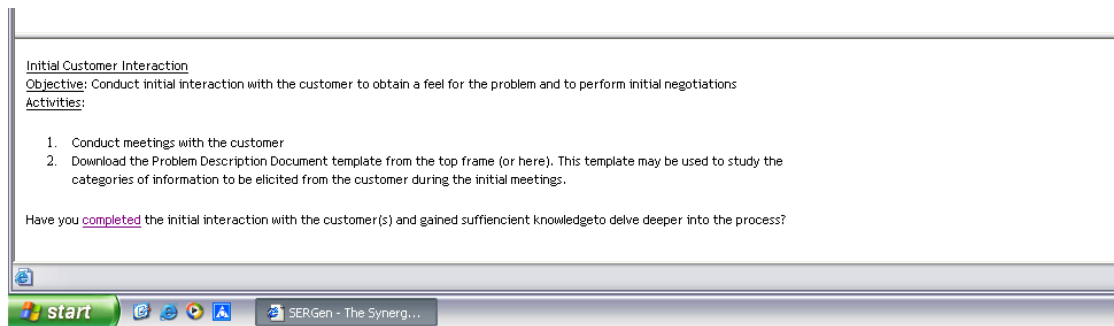


Figure 4.6: “Meaningful” questions (*bottom line*) are asked soliciting information whether a particular task (activity) has been completed or not

4.3.6 Preview Pane

itSRGM also provides an alternate view of the entire requirements generation process. The preview pane enables the user to browse through the entire process, clicking on ‘process bubbles’ to decompose the process into low level activities, or moving the mouse over the bubbles to obtain a brief description of the process. The preview pane does not allow the user to perform any activities listed, but simply provides an intuitive visual representation of the processes. The ‘preview mode’ may be entered upon at the *itSRGM* start page (<http://csgrad.cs.vt.edu/~rsud/Environment>).

4.4 A Synergistic Approach to Software Requirements Generation – *itSRGM* in Action (w/ screenshots)

This section provides a walkthrough, along with complete screenshots for the first process of the Synergistic Model – Conceptual Overview. The current prototype for *itSRGM* implements all of the activities for the first two processes – Conceptual Overview and Problem Synthesis.

As described previously, the implementation comprises 2 modes – the “Preview” mode and the requirements generation mode. The screenshots displayed in this section are from the requirements generation mode.

The tool is implemented in two frames aligned horizontally, one above the other. The top frame provides a visual representation of the model. The frame contents change while progressing through the process, reflecting the current process/activity being performed. The bottom frame contains navigation links, textual guidelines and the objective (Ref. template in section 4.3.1); the top frame depicts where the user currently is within the requirements generation process.

Conceptual Overview

1. (**Figure 4.7**) Conceptual Overview is the first phase in the requirements generation process. We begin by providing a brief description of the current state that is displayed in the top frame. The information in the bottom frame describes the process visualization, being displayed in the top frame.

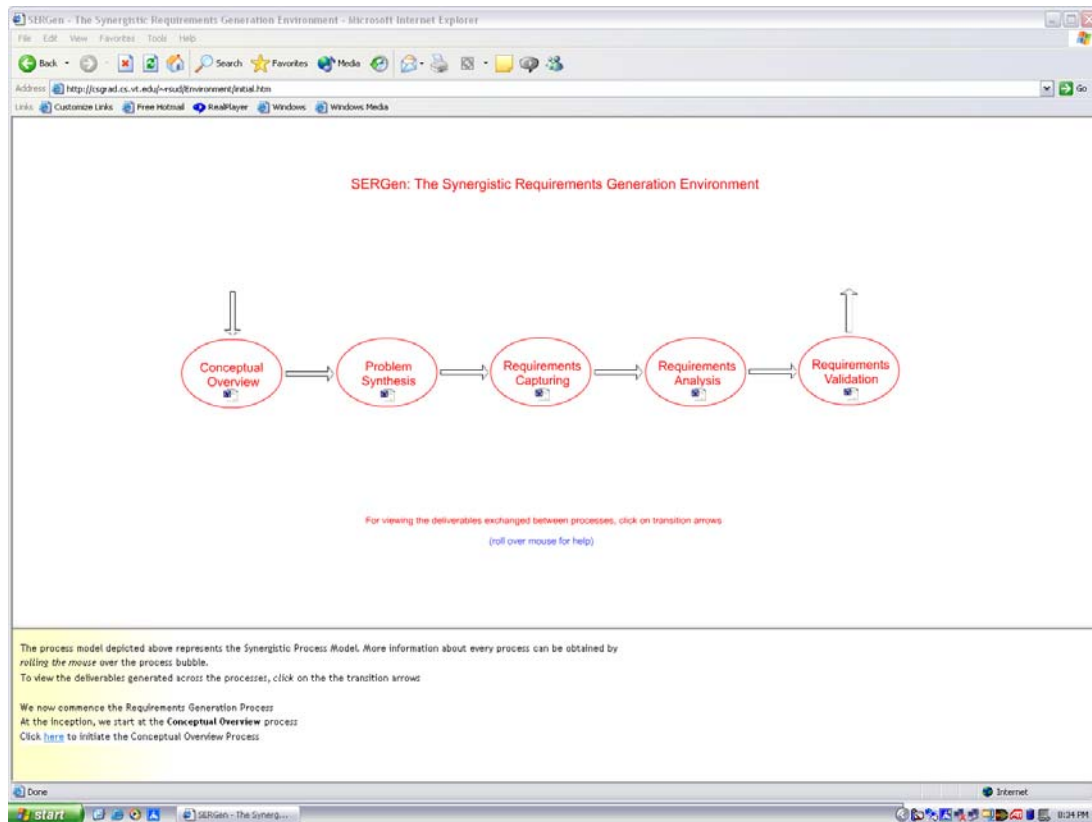


Figure 4.7: The start of the Requirements Generation process, indicating the initiation of the first phase – Conceptual Overview

Following the instruction in the bottom frame, the user clicks on “[here](#)” to move on to the ‘exploded’ view of the Conceptual Overview process. The user is then led to the screen indicated in Figure 4.8.

2. **(Figure 4.8)** The Conceptual Overview process is now “exploded” into two lower level activities comprising: “Initial Customer Interaction” and “Document System Characteristics.” Before starting the “Initial Customer Interaction,” the input source document is required. The lower frame elicits that information from the user. First the type of document is requested, that is, High Level Requirements, Con-ops Document, or Ideas

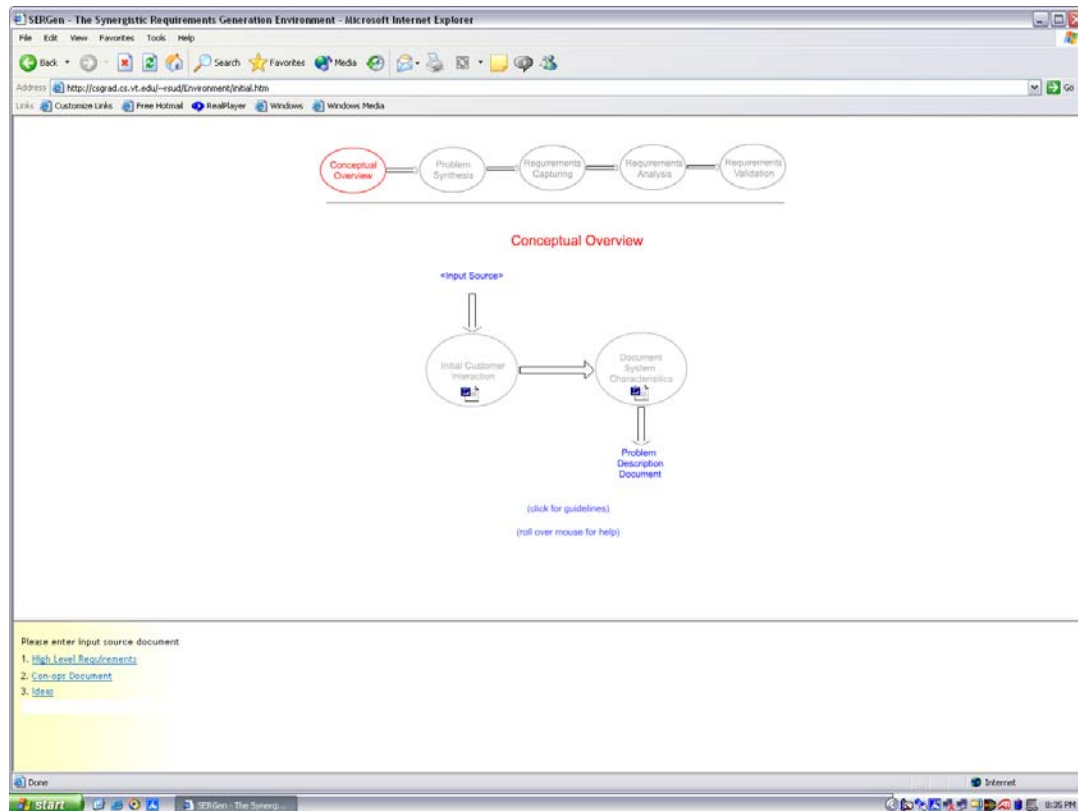


Figure 4.8: Screenshot prompting the user to specify the 'type of the input source document (that was provided by the customer)

Selecting and clicking on the input source document type leads the user led to the next screen (Figure 4.9) requesting the location of the file containing the input.

3. (**Figure 4.9**) Once the type of the document has been identified, the user is prompted to specify the location of the input document. Note the structure that is enforced. The user cannot proceed to the next process unless the destination of the input document is specified. Subsequent prototypes may contain elaborate error checking, verifying textual formatting, to ensure that the document specified is indeed the one that is expected.

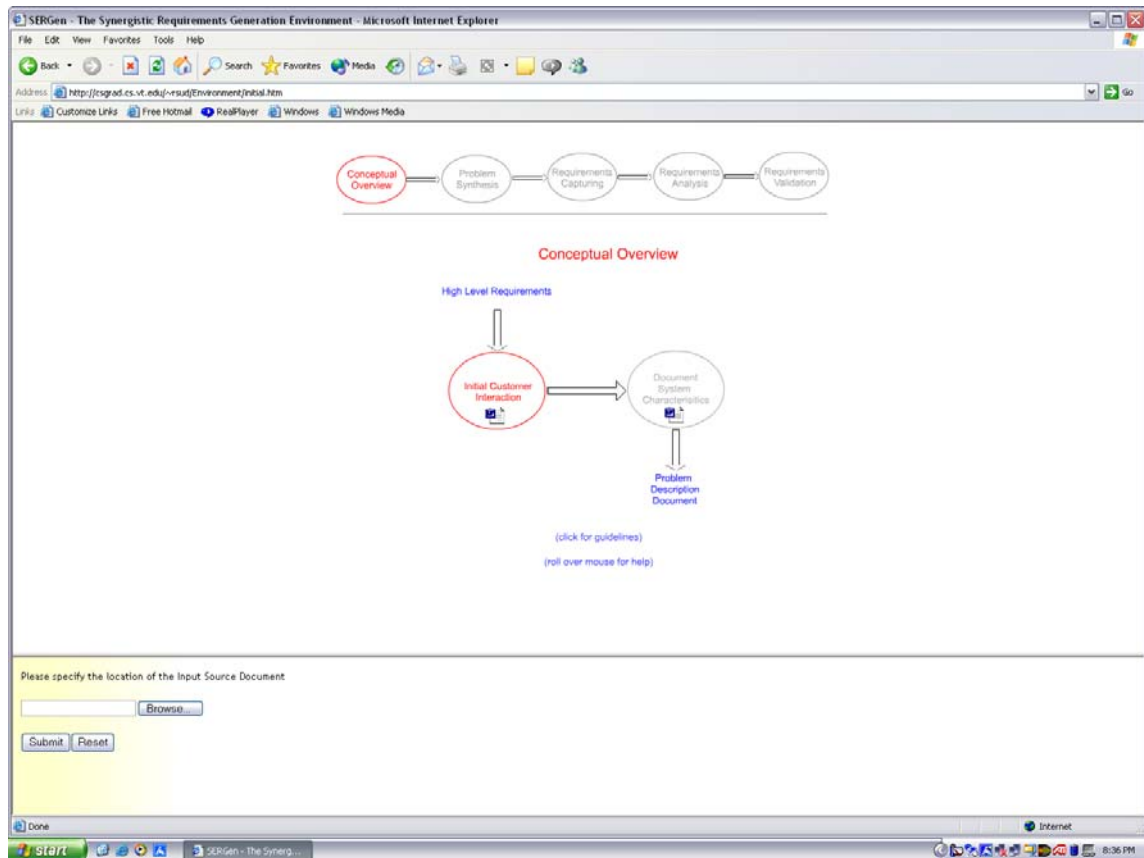


Figure 4.9: Screenshot requesting the user to specify the location of the Input Source Document

The user is expected to ‘browse’ for the input document, and specify its location. Once specified, the user clicks the “Submit” button to proceed to the next screen.

4. (**Figure 4.10**) Once the input document location is specified, the user is led to the first activity defined within the Conceptual Overview phase – “Initial Customer Interaction.” The objective, along with specific guidelines is displayed in the bottom frame, while the top frame provides a visualization of the current position in the process. In the top frame, the process bubble indicating “Initial Customer Interaction” is highlighted, indicating that it is the current process being addressed.

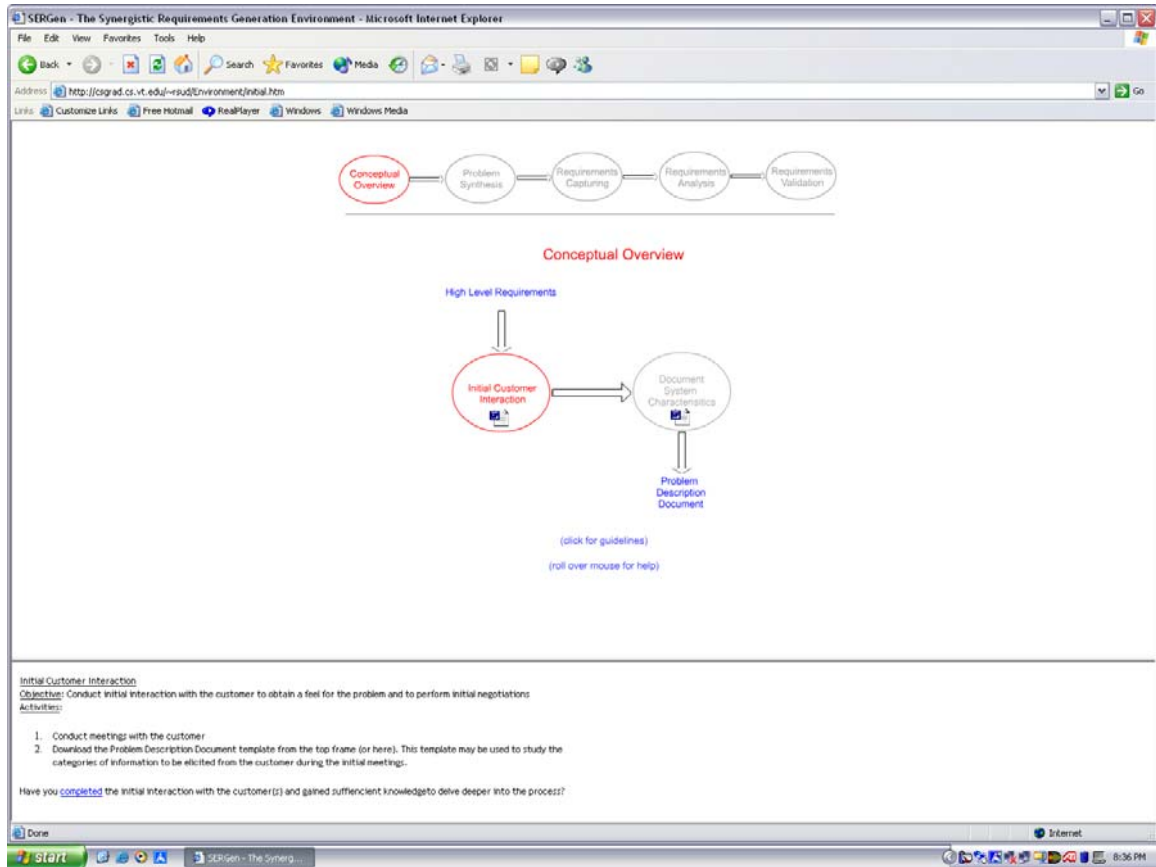


Figure 4.10: Screenshot representing a complete snapshot of the “Initial Customer Interaction” sub-process, complete an objective, guidelines, and a process visualization in the top frame

To proceed to the next process, the user must click on the “completed” hyperlink, thereby responding to the “meaningful” question posed in the lower frame – “Have you completed the initial interaction with the customer(s)...”

5. (**Figure 4.11**) The “Initial Customer Interaction” activity does not expect the user to specify the location or source of any document. The tool assumes the completion of the activity when the user progresses to the next one – “Document System Characteristics.” Once again, an objective and a guideline are specified. The guideline provides a link to the template for the “Problem Description Document.” The user may download the template to ascertain the information required. We do not enforce the use of any template in *it*SRGM, though their use is recommended. Also note the highlighting of the process bubble (in the top frame) associated with “Document System Characteristics,” indicating that it is the current process being addressed.

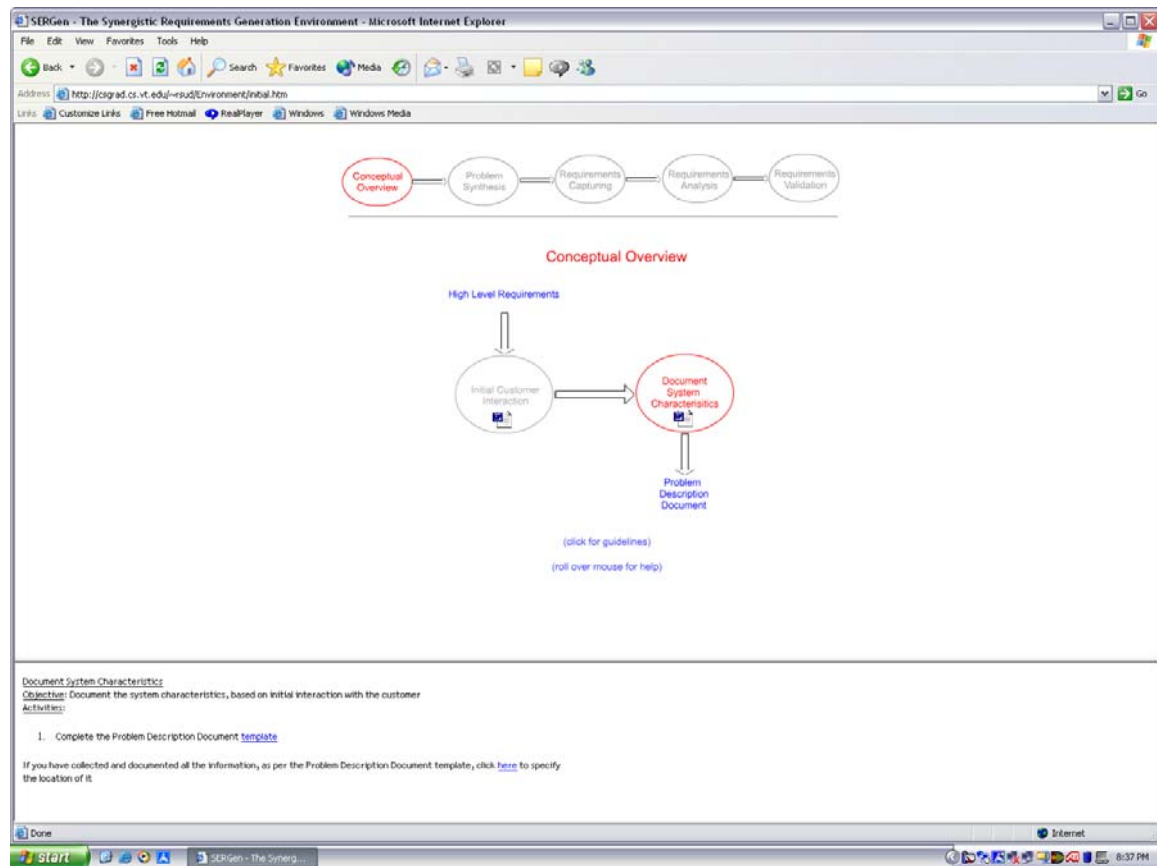


Figure 4.11: Progression to the next sub-process – Document System Characteristics

To proceed to the next process, the user must click on the “[here](#)” hyperlink in the lower frame. Clicking on “[here](#)” implies that the user has documented all the information required in the Problem Description Document.

6. (**Figure 4.12**) After documenting all the information required in the “Problem Description Document” the user is expected to specify the location of the document, in order to progress to the next activity.

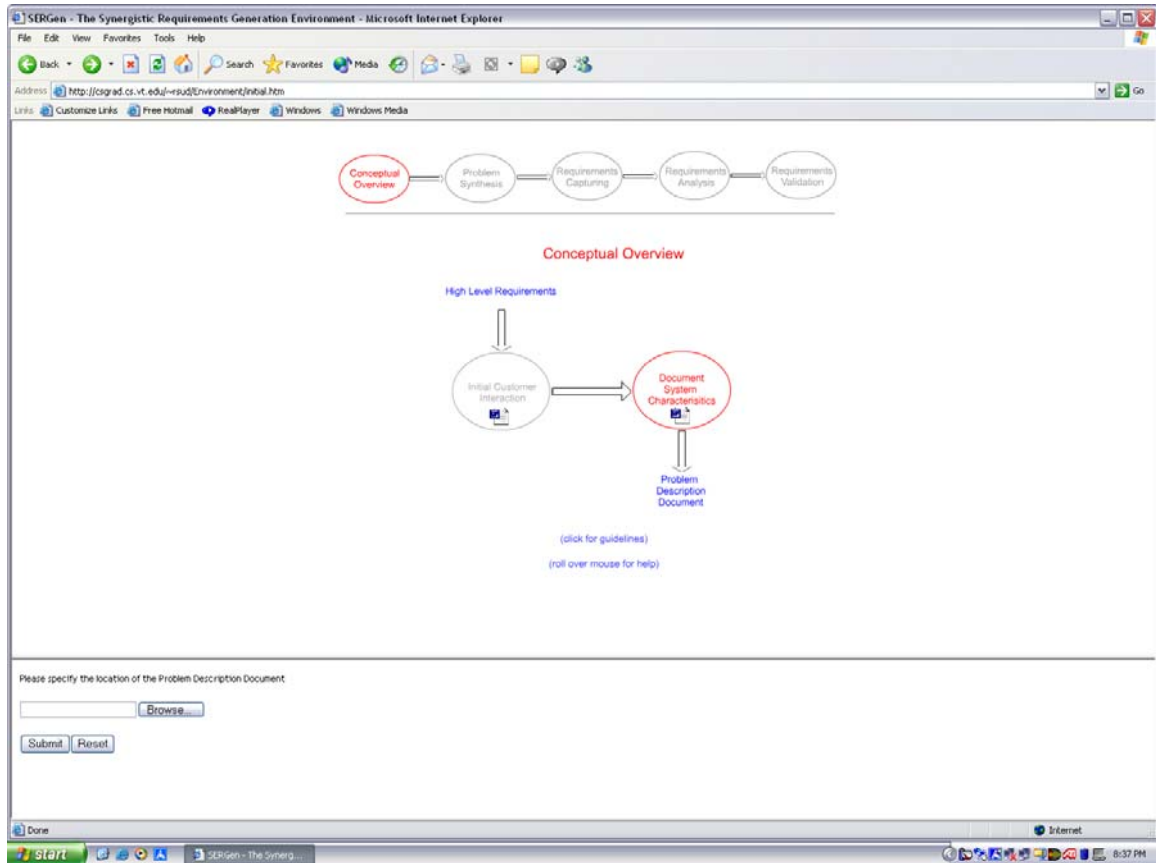


Figure 4.12: Screenshot requesting user to specify the location of the *completed* Problem Description Document

The user is expected to browse for the Problem Description Document, specifying and submitting its location. After completing this, the user is led to the next screen, which indicates the completion of the Conceptual Overview process.

7. (**Figure 4.13**) Since the Problem Description Document is the final artifact, specifying its location completes the Conceptual Overview phase. This is indicated with a screen acknowledging the completion and providing a link to the user to initiate the next process – Problem Synthesis.

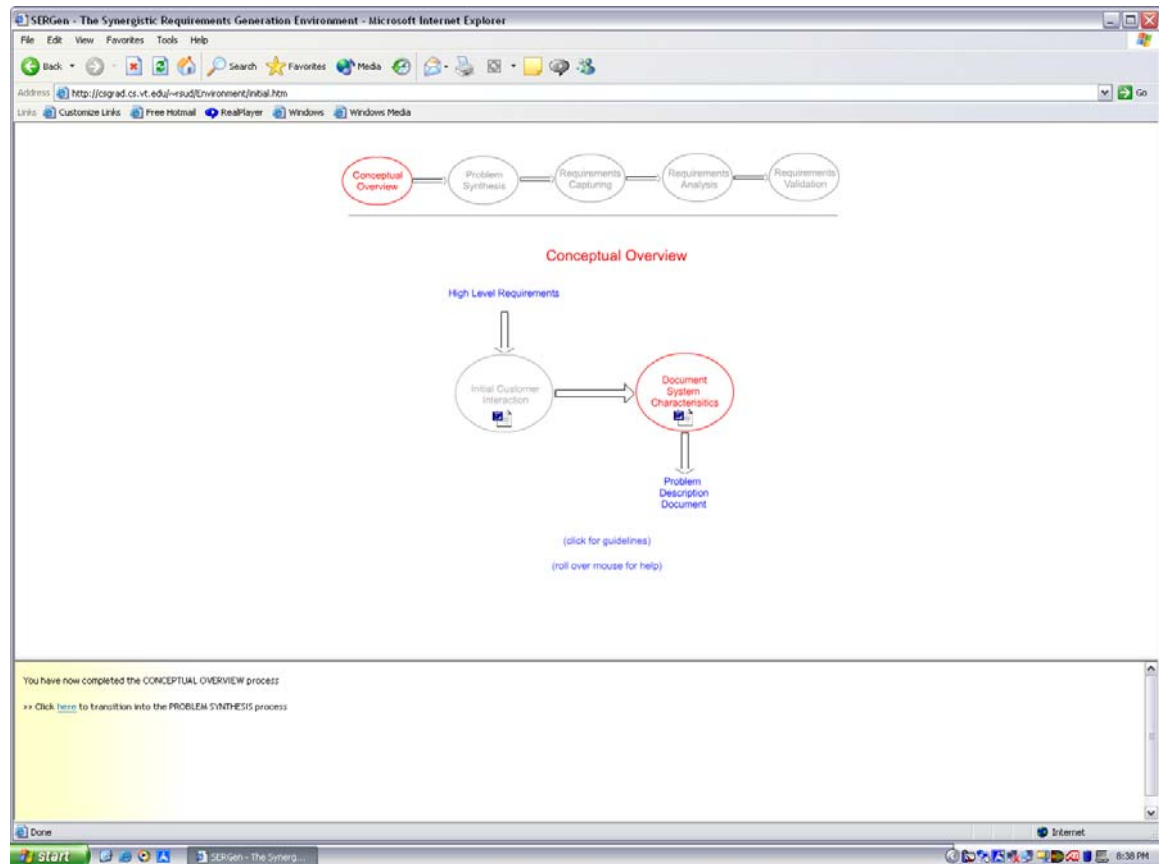


Figure 4.13: Screenshot indicating the completion of the Conceptual Overview process, and initiating transition to the Problem Synthesis process

Similar to the Conceptual Overview, the other processes of the Synergistic Requirements Generation Model will be supported by *it*SRGM. Frames will be provided that will guide the user through Requirements Capturing, Requirements Analysis and Requirements Validation. The current prototype implements the Conceptual Overview and the Problem Synthesis processes. The other processes have been “stubbed” in, but do convey, at the highest level (Figure 4.7), the intent of *it*SRGM.

Chapter 5 Summary and Future Work

5. Summary and Future Work

In the preceding chapters, we have presented a synergistic approach to requirements generation that provides holistic support to requirements engineering. In Chapter 2, we presented the relevant background, which identified the absence and the need for such a process. A study of existing approaches revealed disconnected support for requirements generation, with every approach having a narrow focus of one, two or at most three processes associated with requirements engineering. We then presented the Synergistic Requirements Generation Model, with all its components as an all-encompassing solution approach, along and a tool, *itSRGM*, codifying the model. We now summarize this body of work and present future opportunities for research related to the Synergistic Requirements Generation Model and *itSRGM*.

5.1 Summary

Work on the Synergistic Requirements Generation Model is motivated by the Requirements Generation Model (RGM), [Arthur 1999] and [Groener 2002], which provides a “structured methodological approach to capturing requirements that met the customer intent.” Although focused on requirements elicitation, the RGM provides the structure that was lacking in other approaches at the time. The activities comprising the requirements elicitation process were identified and supplemented with guidelines, protocols and artifacts.

The Synergistic Requirements Generation Model was devised with the intent to provide structure to the overall requirements generation process. The objectives are to (a) provide a participatory approach, which includes inputs from all stakeholders involved with the problem, (b) improve the quality of the requirements by ensuring that the correct problem is addressed, (c) provide artifacts and guidelines aiding the requirements engineer as he/she stepped through the entire process, and (d) codify the model in a form that is easy to comprehend and deploy in any project setting. During conception, care was taken to make the model flexible, with the intent to make it customizable to accommodate any methodology as per specific project needs.

In designing the Synergistic Requirements Generation Model, we studied and evaluated existing requirements engineering processes, and approaches. An in-depth survey of requirements management tools was also conducted to elicit appropriate features. These features, along with concepts we deemed apt, were integrated forming a high level overview of the Synergistic Requirements Generation Model. The high level overview comprised of five processes – Conceptual Overview, Problem Synthesis, Requirements Capturing, Requirements Analysis and Requirements Validation. Conceptual Overview provides the customer and the requirements engineer an introductory overview of the problem, and the solution desired. Problem Synthesis deals with identification, analysis and decomposition of the *real* problem. It also incorporates a process to elicit and analyze the stakeholders' needs. Requirements Capturing elicits requirements from the needs. These requirements are then analyzed from diverse perspectives in the Requirements Analysis process. Finally, the Requirements Validation process validates the requirements with all stakeholders ensuring that the requirements address the real problem and are representative of the customers' expectations.

The SRGM, in its current form, focuses only on the Requirements Generation phase of the Software Development Lifecycle, assuming that all requirements are generated upfront, before software design or implementation. We recognize that this constrains us to fit it in a waterfall-like model, during which requirements are generated upfront. However, we conjecture that the SRGM may be modified to be incorporated in other paradigms as well.

Once devised, these primary processes were split into lower levels of abstraction; each abstraction level providing more detail about the higher-level process. Finally, all processes, sub-processes and low level activities were integrated to form an over arching framework encompassing the entire requirements generation process.

The structure and sequence of the Synergistic Requirements Generation Model evinced a natural flow that could be implemented in the form of a visual interface integrated with tool and artifacts supporting the model. A rudimentary prototype for the Synergistic tool for Requirements Generation *itSRGM* was developed to demonstrate this structure and sequence.

5.2 Contributions

The intent of developing the Synergistic Approach to Requirements Generation, consisting of a model and a tool, was to provide the much-needed direction and structure to the relatively *nascent* field of Requirements Engineering. The approach, by no means is a panacea to solving all problems originating out of requirements, but it is our conjecture that the occurrence of incorrect, incomplete requirements or requirements not adhering to the customer's intent, will be reduced.

The Synergistic Requirements Generation Model provides an innovative method to process modeling. Complex models can be comprehended easily by decomposing them into constituent processes and visually representing the decomposed processes at similar abstraction levels. The visual representations provide an insight into the context of operation, offering varying levels of abstraction, to either probe the details of, or skim through a process.

The benefits of the Synergistic Requirements Generation Model and *itSRGM* are reflective of the need for a participatory approach to the software requirements generation process. The model provides structure, guidance and refinement to processes acknowledged within the requirements engineering realm. The tool, on the other hand, provides an interactive platform to implement and deploy the processes and activities defined within the model.

Specifically, the contents of the *Synergistic Approach* can be summarized as follows:

- The *Synergistic Requirements Generation Model*, decomposing and refining component processes leading to identification of the problem, needs generation, requirements elicitation, analyzing & specifying requirements and requirements verification and validation
- An interactive tool, *itSRGM*, providing an interactive tool structuring and *enforcing* the processes defined within the Synergistic Requirements Generation Model, and consisting of an integrated set of tools and artifacts
- The identification of *Tools and Artifacts*, supplementing *itSRGM*, assisting in communication and documentation of information exchanged between stakeholders

These components may be integrated in any development effort, leading to high quality requirements meeting the customer intent, and aiding planning and scheduling of the requirements generation process; quintessentially, promoting higher levels of satisfaction among all stakeholders.

5.3 Future Work

The intent of the research presented in this thesis is to design an integrated model accounting for all major activities performed in the requirements generation process. A “proof-of-concepts” is developed in the form of an interactive tool codifying the model described above, providing tools and artifacts further augmenting the process. The following three paragraphs outline some possibilities for future development and enhancement efforts regarding the “Synergistic Approach” for requirements generation.

Empirical Evaluation of the Synergistic Requirements Generation Model in a ‘Real World’ setting

Although we conjecture that the Synergistic Requirements Generation Model will enhance the requirements generation process resulting in a more complete, correct specification, a detailed empirical evaluation will help provide a deeper insight into the implementation aspect of the model. An empirical evaluation will judge the effectiveness of the model, possibly highlighting imperfect aspects, if any, of the model. Ideally, the empirical study should be conducted for a real software project, in an organization having troubled experience with software requirements in earlier projects. The model can also be deployed, in the form of a pilot program in any graduate-level Software Engineering-related course, requesting students and hypothetical clients to use the model. This will aid gaining varied perspectives on the effectiveness and the correctness of the processes represented.

Developing a complete prototype for itSRGM, with all processes and sequences elaborated

We have developed a rudimentary prototype representing what we believe will eventually be a complete version of *itSRGM*. The intent is to provide a “proof of concepts,” and a

start for the development of a tool, complete in all respects, providing holistic support to the requirements generation process. The current version is built using Macromedia Flash[®], which primarily is a user interface development platform. An elaborate backend, storing information (requirements) is missing. The tool artifacts and processes are currently “glued” together by hyperlinks. Current tools available in the market can be integrated with tool.

Enhancing the overall model and its decomposition, with additional phases and processes

The Synergistic Requirements Generation Model is designed with extension in mind. An Empirical Evaluation will indeed help uncover inconsistencies in the model. These inconsistencies can be corrected. Additional research will help identify phases and processes that can be integrated in the model. Also, the synergistic approach to representing processes may be expanded to accommodate more aspects of the software development lifecycle.

Adapting the Model to Other Software Development Paradigms

The current form of the SRGM assumes that all requirements are to be generated upfront, before system design or implementation. This makes it more suitable to be applied in a Waterfall-like development paradigm. As a part of the future work, the SRGM may be adapted to ‘fit’ to more recent development paradigms, such as incremental and evolutionary approaches. For example, considering the spiral development paradigm [Boehm 1988], the ‘Conceptual Overview’ and the ‘Problem Synthesis’ phases of the SRGM may be performed upfront, at the start of the process. This will help enable the *real* problem at hand, for which a solution is to be developed. Subsequently, the requirements capturing, analysis and validation activities may be instantiated during each iteration of the ‘spiral.’

References

- Akao, Y., 1990, *Quality Function Deployment: Integrating Customer Requirements into Product Design*, Productivity Press: Cambridge.
- Alford M.W., 1977, *Management of Requirements Development Using SREM Technology*, Huntsville, AL.
- Arthur, J.D. and Markus K. Groener, 1999, *An Operational Model Supporting the Generation of Requirements that Capture Customer Intent*, Proceedings of the Pacific Northwest Software Quality Conference, Portland OR, October 1999, pp. 286-302.
- Beck K., 1999, *Extreme Programming Explained: Embrace Change*. New York, NY: Addison-Wesley.
- Bell T. E. , T. A. Thayer, 1976, *Software requirements: Are they really a problem?*, Proceedings of the 2nd international conference on Software engineering, p.61-68, October 13-15, 1976, San Francisco, California, United States.
- Boehm, B.W., 1988, *A Spiral Model of Software Development and Enhancement*, IEEE Computer, Vol. 21, No. 5, May 1988, pp. 61-72.
- Booch, G., 1986, *Object-Oriented Development*, IEEE Transactions on Software Engineering, Vol. 12, No. 2, February 1986, pp. 221 ff.
- Brackett J.W., 1990, *Software Requirements*, SEI Curriculum Module SEI-CM-19-1.2, January 1990.
- Brooks F.P., Jr., 1987, *No silver bullet: essence and accidents of software engineering*, Computer, v.20 n.4, p.10-19, April 1987.
- Brooks F. P. Jr., 1995, *The Mythical Man Month*, Anniversary Edition, Addison Wesley Publishing Co.
- Cameron, J.R., 1986, *An Overview of JSD*, IEEE Transactions on Software Engineering, vol. 12, no. 2, 222-240.
- Carmel E., R.D. Whitaker, and J.F. George, 1993, *PD and Joint Application Design: A Transatlantic Comparison*, Communications of the ACM, Vol. 36, No. 4, June 1993.

- Carter, R.A.; Anton, A.I.; Dagnino, A.; Williams, L., 2001, *Evolving beyond requirements creep: a risk-based evolutionary prototyping model*, Requirements Engineering, 2001. Proceedings, Fifth IEEE International Symposium on, Vol., Iss., 2001, pg. 94-101.
- Charette, Robert N., 1990, *Applications Strategies for Risk Analysis*. New York: McGraw-Hill.
- Checkland, P.B., 1989, *Soft Systems Methodology*, In: Rosenhead, J. (ed.): Rational Analysis for a Problematic World, John Wiley & Sons: Chichester, pg. 71-100.
- Chin K.F., *A JAD Experience*, 1995, Proceedings of the 1995 SIGCPR Conference on Supporting Teams, groups, and leaning inside and outside the IS function reinventing IS, pg. 235-236, ACM Press, New York.
- Conklin J.E.; Begeman, M.L., 1988, *gIBIS: A Hypertext Tool for Exploratory Policy Discussion*. ACM Transactions on Office Information Systems, vol. 6, no. 4, 303-331.
- Davis, A.M., 1993, *Software Requirements: Objects, Functions, & States*, Prentice-Hall, Upper Saddle River, New Jersey.
- Davis A., R. Fairley, E. Yourdon, 1999, *Software Product Planning*, Omni-Vista White Paper #99-002, <http://hpsearch.uni-trier.de/hp/a-tree/d/Davis:AI.html>, October 1999.
- Gause D., and G. Weinberg, 1989, *Exploring Requirements: Quality Before Design*, Dorset House publishing, New York.
- Glass, R., 1998, *Defining Quality Intuitively*, IEEE Software, pp. 103-104,107, May 1998.
- Goguen J.A., C. Linde, 1993, *Techniques for Requirements Elicitation*, In Proceedings of the IEEE International Symposium on Requirements Engineering, San Diego, CA, pg. 152-164. IEEE, New York, January 1993.
- Groener Markus, 2002, Ph.D. Dissertation, Virginia Tech ETD Collection, <http://scholar.lib.vt.edu/theses/available/etd-05232002-234024/>
- Herlea, D.E., Jonker, C.M., Treur, J., and Wijngaards, N.J.E., 1999, *A Formal Knowledge Level Process Model of Requirements Engineering*, In Proceedings of the 12th International Conference on Industrial and Engineering Applications of AI and Expert Systems, IEA/AIE'99.
- Hofmann Hubert F., 1993, *Requirements Engineering: A Survey of Methods and Tools*, 1993.

- Hofmann, H.F., Lehner, F., 2001, *Requirements engineering as a success factor in software projects*, IEEE Software, Vol.18, Iss.4, Jul/Aug 2001, Pages: 58-66.
- IBM Research, 2003, <http://www.research.ibm.com/hyperspace/>
- Jones, Capers, 1994, *Assessment and Control of Software Risks*, Englewood Cliffs, NJ: PTR Prentice Hall.
- Lauesen S., 2002, *Software Requirements: Styles and Techniques*, Addison Wesley Publishing Co.
- Lawrence, B., 1996, *Unresolved Ambiguity*, American Programmer 9(5): pg. 17-22.
- Leffingwell D, D. Widrig, 2000, *Managing Software Requirements: A Unified Approach*, Addison Wesley Publishing Co.
- NASA, 1997, *Writing Effective Requirements Specifications*, Software Technology Conference, Utah, April 1997.
http://satc.gsfc.nasa.gov/support/STC_APR97/write/writert.html.
- Ohnishi, A., 1995, *CARD: An environment for software requirements definition*, Software Engineering Conference, 1995. Proceedings, 1995 Asia Pacific, Vol., Iss., 6-9 Dec 1995, pg. 420-429.
- Pressman R.S., 2001, *Software Engineering: A Practitioner's Approach*, 5th ed. New York, NY: McGraw- Hill, 2001.
- Royce, W.W., 1970, *Managing the Development of Large Software Systems: Concepts and Techniques*, Proceedings of the Western Electronic Show and Convention (WesCon), Los Angeles, August 1970, pp. 1-9 (Reprinted in the Proceedings of 9th International Conference on Software Engineering, March 1987, Monterey CA, pp. 328 – 338.)
- Sidky, A.S., Sud, R.R. Bhatia, S, and Arthur, J.D., 2002, *Problem Identification and Decomposition within the Requirements Generation Process*, 6th World Multiconference on Systems, Cybernetics, and Informatics (SCI 2002), Vol. VIII, July 2002, Orlando FL, pp. 333-338.
- Smith, T.J., 1993, *READS: A requirements engineering tool*, Requirements Engineering, 1993, Proceedings of IEEE International Symposium on, Vol., Iss., Pages:94-97, 4-6 Jan 1993.
- Sommerville I., 1996, *Software Engineering*, 5th ed. Reading, Mass.: Addison Wesley Publishing Co.

Sommerville, Ian, and, Pete Sawyer, 1997, *Requirements Engineering: A Good Practice Guide*. Chichester, England: John Wiley & Sons.

Standish Group, 1995,

http://www.standishgroup.com/sample_research/chaos_1994_1.php.

Stokes, David Alan, 1991, Requirements Analysis, Computer Weekly Software Engineer's Reference Book, pp. 16/3-16/21

Sud Rajat R., and, J.D. Arthur, 2003, Requirements Management Tools: A Quantitative Assessment, Technical Report TR 03-10, Department of Computer Science, Virginia Tech.

Systems Designers, 1985, *CORE: The Method*. Systems Designers Scientific: Camberly.

Weigers K.E., 1999, *Automating Requirements Management*,

http://www.processimpact.com/articles/rm_tools.html.

Weigers K.E., 2001, *Software Requirements*, Microsoft Press, 2001.

Zahniser R.A., 1990, *Building Software in Groups*, American Programmer, vol. 3, nos. 7-8, July-August, 1990.

Appendix A: Requirements Generation Model, Artifacts and Templates

A.1 Overview

The Synergistic Requirements Generation Model, under development is novel and provides the requirements engineering process the much-needed structure that has been lacking. The visual representation of the tool provides guidelines and the context in which attendant activities progress. A prescribed process is also embodied with sets of tools and artifacts. For example, a text editor provides the means to document the requirements. An automated requirement measurement tool¹⁰ is then employed to parse the requirements specification and ensure consistency and unambiguity. The process will be further populated with tools such as *email managers*, *hierarchy generators*, *schedulers*, etc. The operation of all the tools is under the auspices of the framework outlining the process. Section 2 of this Appendix provides tabulated description of the process defined by *itSRGM*. Section 3 describes the supporting tasks and artifacts.

A.2 Synergistic Requirements Generation Model – *Tabulated Process Descriptions*

This section provides a tabulated summary of all “major” processes of the Synergistic Requirements Generation tool, along with inputs, outputs, participants and their roles, and the tasks to be performed.

¹⁰ NASA ARM - <http://satc.gsfc.nasa.gov/tools/arm/>

A.2.1 Conceptual Overview

Process: Generate Conceptual Overview		Process ID: 1.0
Description: The Requirements Generation Process begins with a Conceptual Overview of the proposed system. This process deals with initial interaction/negotiation between the customer and the project management		
Entry Criteria/Inputs: • High Level System Requirements • Concept of Operations Document • Ideas		Exit Criteria/Outputs: • Problem Description Document • System Conceptual Overview
Participants: • Customer – Provides overview of the problem and the context • Senior Management – Negotiation • Requirements Engineer – Gains an understanding of the symptoms and type of problem • Systems Engineering – Provides high level system requirements/context of the overall system, of which the software is a part • Project Management – Prescribes schedule and other constraints		
References: http://www.chambers.com.au/Sample_p/s_conops.htm		
Tasks: 1. Initial Customer Interaction 2. Document System Characteristics		

PROCESS TASKS:

1. **Initial Customer Interaction:** agree on introductory goals, constraints, and acceptance criteria
 - a. Identify primary stakeholders
 - b. Preliminary Negotiation on schedule and cost constraints
 - c. Preliminary Introduction to the problem at hand
 - d. Review existing documentation, i.e. High Level System Requirements/Con-ops Document/Ideas
2. **Document System Characteristics** - formalize proceedings of the interaction in the form of a System Conceptual Overview, called the Problem Description Document (Table 3.2)

A.2.2 Problem Synthesis

Process: Problem Synthesis		Process ID: 2.0
Description: Problem Synthesis deals with the recognition of the real problem at hand. Activities performed diagnose the observable symptoms to determine the source of the problem. This process is imperative as it is crucial to identify the problem, gain an understanding of that problem so that the correct solution is devised addressing the actual problem		
Entry Criteria/Inputs: Problem Description Document		Exit Criteria/Outputs: - Problem Statement - Set of Needs/Customer Requirements - In-depth Understanding of the real problem
Participants: - Customer – Provides input in the form of observable symptoms/desirable functionality - Requirements Engineer – Conducts Ethnographical field studies, Root Cause Analysis, etc. to determine the real problem - System Engineering – Provides high level requirements from which software requirements are to be elicited - Project Management – Provides Con-ops Document		
References: [Leffingwell 2000] [Davis 1993]		
Tasks: - Requirements Engineer Indoctrination - Problem Analysis - Context and Constraints Analysis		

PROCESS TASKS:

- Requirements Engineer Indoctrination:** educate the Requirements Engineer of the context in which the system is to be developed/deployed
 - Conduct Initial Meetings with stakeholders
 - Develop plan of indoctrination
 - Conduct customer walkthroughs, explaining the problem
 - Conduct ethnographical field studies
- Problem Analysis:** activities conducted to gain an understanding of the problem
 - Problem Identification
 - Problem Decomposition
 - Context and Constraints Analysis
- Needs Generation:** activities conducted to generate needs from the problem elements derived as an outcome of the Problem Decomposition process. The following activities are conducted within an iterative process
 - Needs Elicitation Meeting
 - Analysis
 - Evaluation

Process: Requirements Engineer Education	Process ID: 2.1
Description: The objective of educating the Requirements Engineer is to provide them the best possible information about the current system, how it accounts for the customer's needs, as well as the context and environment in which the new system will be deployed	
Entry Criteria/Inputs: • None	Exit Criteria/Outputs: • Domain knowledge
Participants: • Customer – Presents the current system (if one exists), including the environment in which it is installed, and an explanation of what needs are currently fulfilled by the system • Requirements Engineer – Conducts Ethnographical field studies, Root Cause Analysis, etc. to determine the real problem	
References: [Arthur 1999]	
Tasks: 1. Indoctrinate the Requirements Engineer	

PROCESS TASKS:**Customer**

1. Presentation of the current system, if one exists
2. Introduction to the needs that the proposed system must address
3. Motivating the need for a change/new system

Requirements Engineer

1. Conduct ethnographical field studies and observations, related to the environment in which the new system is to be deployed
2. Encourage customer and users to bring forth ideas and needs

Process: Problem Analysis		Process ID: 2.2
Description: Problem Analysis is devoted to identifying and understanding the problem. Artifacts produced formally document the problem for later reference (verification and validation activities)		
Entry Criteria/Inputs: • Problem Description Document		Exit Criteria/Outputs: • Problem Statement • Problem Elements • Context Diagram • Constraints Document
Participants: • Customer • Requirements Engineer • End Users		
References: [Leffingwell 2000]		
Tasks: 1. Problem Identification 2. Problem Decomposition 3. Context and Constraints Analysis		

PROCESS TASKS:

1. Problem Identification
2. Problem Decomposition
3. Context and Constraints Analysis

These sub-processes are elaborated in greater detail in the tables overleaf.

Process: Problem Identification	Process ID: 2.2.1
Description: Problem Identification comprises a series of activities, performed iteratively, to identify the problem at hand. The output is a Problem Statement, specifying the real problem. In case the problem exists prior to entering this process, its function will be to verify the problem, to ensure that it is the <i>real</i> problem	
Entry Criteria/Inputs: • Problem Description Document (from Conceptual Overview)	Exit Criteria/Outputs: • Problem Statement • In-depth Understanding of the real problem
Participants: • Customer – Provides input in the form of observable symptoms/desirable functionality • Requirements Engineer – Conducts Ethnographical field studies, Root Cause Analysis, etc. to determine the real problem • End Users	
References: [Leffingwell 2000]	
Tasks: 1. Initial Set of Meetings 2. Develop Stakeholder Profiles 3. Ethnographical Field Studies/Observations 4. Root Cause Analysis 5. Problem Verification (with stakeholders)	

PROCESS TASKS:

- 1. Initial Set of Meetings:** conduct initial set of meetings to obtain information about the stakeholders, environment and expectations of various stakeholders
- 2. Develop Stakeholder Profiles:** develop stakeholder profiles and classify according to project interest. For example, managers, end users, senior management, development team, project management etc.
- 3. Ethnographical Field Studies/Observations:** conduct ethnographical field studies and observe work place and environment where the proposed system is to be deployed. In case of an existing system, observe functioning and shortcomings/problems
- 4. Root Cause Analysis:** once a set of potential problems have been identified, perform root cause analysis to determine the most pertinent causes
- 5. Problem Verification (with stakeholders):** it is imperative that the proposed problem be presented to the customer and a mutual agreement be sought

Process: Problem Decomposition		Process ID: 2.2.2
Description: Problem Decomposition deals with the decomposition of the problem identified earlier. Decomposition is performed with the intent to gain a deeper understanding of the problem. The problem is decomposed into constituent elements. Needs are subsequently generated for each of these problem elements.		
Entry Criteria/Inputs: • Problem Statement (duly verified)		Exit Criteria/Outputs: • Problem Elements
Participants: • Customer • Requirements Engineer – Conducts meetings with stakeholders, to decompose the problem • End Users – provide necessary detail and domain knowledge required in the decomposition process		
References: [Leffingwell 2000]		
Tasks: 1. Conduct Meetings with Stakeholders 2. Identify Key Problem Areas		

PROCESS TASKS:

1. **Meetings with Stakeholders:** conduct meetings with stakeholders to modularize the problem and decompose into constituent elements, identifying and classifying components. For example, categories can be user interaction, backend, etc.
2. **Identify Key Problem Areas:** Cause-Effect analysis, Root Cause analysis and Pareto analysis help identify the problem areas

Process: Context and Constraints Analysis		Process ID: 2.2.3
Description: Context Analysis determines the boundary of the problem and this confines needs generation within these boundaries. Constraints Analysis identifies the degree of freedom that the solution space should possess.		
Entry Criteria/Inputs: • Problem Statement		Exit Criteria/Outputs: • Context Diagrams • Constraints Document
Participants: • Requirements Engineer – Conducts Ethnographical field studies, Root Cause Analysis, etc. to determine the real problem • End User – Provides relevant domain knowledge • Senior Management – Provides information about the constraints to be imposed on the system		
References: [Leffingwell 2000]		
Tasks: 1. Identify System Role with respect to Organization 2. Identify Actors interacting with the System 3. Elicit Constraints on the System		

PROCESS TASKS:

1. **Identify System Role with respect to Organization**
2. **Identify Actors interacting with the System**
3. **Elicit Constraints on the System**

Process: Needs Generation		Process ID: 2.3
Description: The Needs Generation Process translates the problem elements into representative needs/customer requirements. Each problem element may have multiple needs associated with it.		
Entry Criteria/Inputs: • Problem Statement • Problem Elements • Context Diagrams • Constraints Document		Exit Criteria/Outputs: • Set of Needs/Customer Requirements
Participants: • Customer • Requirements Engineer • End User		
References: [Leffingwell 2000]		
Tasks: - Needs Generation Meetings - Needs Analysis - Evaluation		

PROCESS TASKS:

- Needs Generation Meetings:** elicitation meetings are conducted in accordance with the protocols for conducting “structured meetings.” Interaction is facilitated between representative users and the requirements engineer. The following approaches may be adopted:
 - Viewpoints – a collection of information about a system or a related problem, environment or domain, which is collected from a particular perspective (end-user perspective, developer perspective, etc.)
 - Context Free Questions
 - Interviews/Questionnaires
- Needs Analysis:** needs generated from the elicitation meetings are related and analyzed with respect to the constraints and context
 - Convergence
 - Prioritization
 - Conflict Resolution
- Evaluation:** needs are evaluated to verify the right information and decide whether another iteration is required for the particular problem element

A.2.3 Requirements Capturing

Process: Requirements Capturing		Process ID: 3.0
Description: Requirements Capturing deals with the generation of <i>actual</i> requirements appearing, prospectively, in the Software Requirements Specification.		
Entry Criteria/Inputs: • Set of Needs • Context and Constraints Document		Exit Criteria/Outputs: • Requirements (unvalidated)
Participants: • Requirements Engineer/ Software Engineer • End Users • Customer		
References: [Arthur 1999] [Stokes 1991] [NASA 1997] - IEEE Std 830-1993, Recommended Practice for Software Requirements Specification, December 2, 1993		
Tasks: - Customer Indoctrination - Requirements Elicitation Meeting - Local Analysis - Evaluation		

PROCESS TASKS:

1. **Customer Indoctrination:** educate the customer about the process model and the method adopted of eliciting and generating requirements
 - a) Educate customer/end user of conduction of elicitation meetings
 - b) Educate customer about the requirements generation process
 - c) Explain roles, tasks and responsibilities of customer throughout the generation process
2. **Requirements Elicitation Meeting:** conduct the elicitation meeting as per the methodology (JAD/PD) selected
 - d) Identify Meeting Participants
 - e) Set Meeting Agenda
 - f) Dispatch Meeting Notification
 - g) Conduct Elicitation Meeting
 - h) Document Minutes and Proceedings
3. **Local Analysis:** assign one or more of the following attributes to every requirement generated as a result of the elicitation meetings (transform candidate technical requirements into informal requirements)
 - a) Assign Attributes
 - Assign Status
 - Assign Priority
 - Estimate Effort
 - Assign Risk
 - b) Rank Requirements

4. **Evaluation:** perform assessment of requirements to verify addressing of quality attributes, to uncover inconsistencies and redundancies, verify all technical requirements have been incorporated and that they are testable
 - a) Check Quality Attributes (Correctness, Unambiguous, Complete, Consistent, Ranked, Verifiable, Modifiable, Traceable, Understandable)
 - b) Determine if additional iteration is required

A.2.4 *Requirements Analysis*

Process: Requirements Analysis		Process ID: 4.0
Description: Requirements Analysis incorporates processes that ensure the correct software product is being built. A 'correct' software product has the "correct mix of price, cost, and feature." Based on the requirements generated from the Requirements Capturing process, certain 'global' activities are performed that evaluate the 'fate' of the software product, from various standpoints.		
Entry Criteria/Inputs: • Software Requirements (unvalidated)		Exit Criteria/Outputs: • Formatted Software Requirements Specification (SRS) (unvalidated)
Participants: • Requirements Engineer • Software Engineer • Development Team		
References: [Davis 1999]		
Tasks: 1. Risk Analysis 2. Cost and Schedule Estimation 3. Price and Market Analysis 4. Feasibility Analysis 5. Requirements Specification		

PROCESS TASKS:

1. Global Analysis

- Risk Analysis** - the Local Analysis activity assigned a risk factor in the form of a rank to the risk associated with each requirement. This is now weighed in collectively with all the requirements generated so far
 - Cost and Schedule Estimation** - determine the effort and time required to implement potential features
 - Price and Market Analysis** - determine optimal price to charge to customers, or market demand (identify customer types, buying characteristics, marketing, etc.), in case of shrink-wrapped product
 - Feasibility Analysis** - determine the feasibility and the profitability associated with the software product. Based on this analysis, a 'go-no go' decision is to be taken
- Requirements Specification** - All the analysis factors, determined above, associated with the requirements, are included in the requirements specification document.

Global Analysis

The Global Analysis set of activities represent a **suite of decisions** that *must* be taken, as and when requirements/features are generated, and attributes, such as those listed above, are associated with the requirements. The Global Analysis have to be captured from 3 different perspectives:

- o Development
- o Marketing and Sales
- o Financial

Development Perspective

- o Can the selected features be built?
- o By when can the selected features be built?
- o How many resources are required to build the selected features?
- o What kinds of resources are required to build the selected features?
- o How much risk can the organization tolerate?
- o What are the ramifications of being late or being over-budget?

Marketing and Sales Perspective

Note: This perspective will be included only when the software development effort deals with the development of a shrink-wrapped product.

- o How big is the market potential for a product with the selected features?
- o At what price can we sell a product with the selected features?
- o When does the customer want the product?
- o How does the proposed product compare with the competition?

Financial Perspective

- o How will the use of the product save (or earn) the company each time it is used?
- o How much is the customer willing to pay for the software development effort?

Capturing and documenting these perspectives is critical to the progress of the development effort considering the competitiveness in the software industry.

A.2.5 Requirements Validation

Process: Requirements Verification and Validation		Process ID: 5.0
Description: The purpose of Verification and Validation is to provide assurance that the artifacts (SRS) are truly representative of the customer requirements/needs. This is accomplished by presenting the formal SRS to the stakeholders, actively involving the stakeholders in formal technical reviews, and development of prototypes, thereby facilitating agreement among the stakeholders		
Entry Criteria/Inputs: Formatted Software Requirements Specification (SRS) (unvalidated)		Exit Criteria/Outputs: Software Requirements Specification (SRS)
Participants: - Requirements Engineer - Customer - Senior Management - Project Management		
References: - Requirements Management Guidebook [Leffingwell 2000] http://www2.ics.hawaii.edu/~johnson/FTR/presentations http://www.asq-software.org/mhm/ftr.htm		
Tasks: - Assessment of SRS to verify quality attributes - Traceability to customer requirements/needs - Conduct Formal Technical Reviews - Develop System Prototypes - Establish a Baseline for the SRS		

PROCESS TASKS:

- 1. Verification:** ensure the artifacts produced meet non-functional requirements
 - Perform assessment of requirements to verify addressing of quality attributes, to uncover inconsistencies and redundancies, verify whether all technical requirements have been incorporated and that they are testable
- 2. Validation:** approve baselined requirements reflecting acceptance criteria
 - Formalize commitment by all stakeholders to baselined requirements
 - Obtain customer approval
 - Obtain approval to proceed to the next phase of development – System Design
 - Verify traceability to (higher-level) customer requirements/needs
 - Conduct Formal Technical Reviews to facilitate high degree of customer commitment and acceptance
 - Facilitate agreement on requirements between project management and the customer
 - Establish a requirements baseline by placing artifacts under configuration management

Formal Technical Reviews

A Formal Technical Review (FTR) is a 'team effort' to eliminate problems in the deliverables through group reviews. This section provides a brief overview of the importance of conducting FTRs.

Reasons to conduct FTRs:

- o To ensure that the requirements specification is complete and unambiguous
- o To ensure that the requirements meet the customer intent
- o To detect and correct errors and discrepancies early in the development stage

One or more of the following approaches may conduct FTRs:

- o Peer Reviews
- o Inspections (by upper management or independent agencies)
- o Walkthroughs

The method of conducting an FTR Meeting is similar to that of the Structured Meeting (4.1). The objectives, however differ

- o Defect Removal in the requirements specification
- o Assess quality of the requirements specification and the process

System Prototypes

A software requirements prototype is a partial implementation of a software system, built to help developers, users and customers better understand the requirements of the system

A prototype, built by the developer, serves the following purposes:

- o Used to obtain customer approval/confirmation that the developer understands the requirements and the intent of the customer
- o Used to elicit more requirements

The customer to help communicate requirements to the developer may also construct a prototype

Guidelines for developing Prototypes:

1. The well-known and well-understood requirements need not necessarily be prototyped
2. The prototype should be built in a manner that consumes the least amount of resources

After being built, the prototype should be 'exercised' in the user environment eliciting comments, improvements and/or additional requirements

A.3 Requirements Management Tools Supporting *itSRGM*

This section provides a list of tools being incorporated in the requirements generation platform. It must be noted that these tools are not just applicable to *itSRGM*, but can be generically applied to any requirements management effort.

Hierarchy Generator

The first activity performed while organizing a requirements elicitation meeting is participant selection. This selection varies depending upon the methodology employed for conducting the elicitation meeting. For example, Joint Application Development¹¹ (JAD) and Participatory Design¹² (PD) advocate different approaches to selecting participants. JAD promotes the selection of a cross-section of a hierarchy of users. It advocates that participants should preferably represent all ‘user classes.’ On the other hand, PD encourages the selection of participants from the same *strata*. According to PD, this helps participants *shed inhibitions* and engage more actively in the proceedings, without having the fear of being evaluated by seniors. For the advantages and disadvantages of each methodology, the reader is advised to refer to the footnote at the end of this page.

For a requirements engineer who is not a part of the organization, selecting participants can be a daunting task, especially in larger projects where requirements are to be elicited from a large populace. Remembering the name, contact information, role and designation of each stakeholder can be difficult.

The Hierarchy Generator helps in managing information about stakeholders and selecting representative participants. Referring to Figure A.1, each node in the tree represents the information about the stakeholder, including the contact details, role and other pertinent information. The corresponding level of the tree represents the designation of the stakeholder in the organization. The requirements engineer enters information about each stakeholder and a tree-like structure will be generated automatically.

¹¹ For more details refer [Chin 1995]

¹² For more details refer [Carmel 1993]

Similarly, the selection criterion for the PD methodology is depicted in Figure A.3.

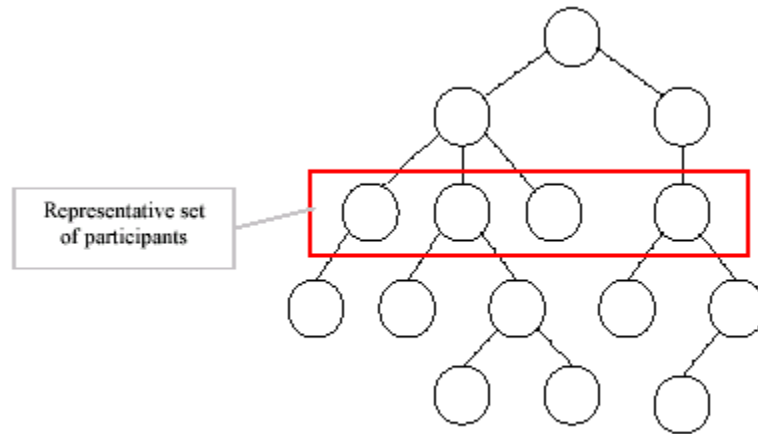


Figure A.3: Participant Selection as per the PD Methodology

The Hierarchy Generator automates the selection of the representative set of participants and sends out the elicitation meeting notification to every participant.

Scheduler

The Scheduler is responsible for *charting* the course of activities to be performed to ensure an effective requirement generation process. An underlying framework supports the interactive tool. This framework dictates the sequence of activities performed and accounts for the need of an additional iteration. Simply put, the framework is responsible for scheduling and planning all activities.

The Scheduler displays the timeframe within which these activities are to be performed, as shown in Figure A.4. The front-end of the Scheduler consists of a calendar that displays what activity is to be performed and when. The Scheduler is *dynamic*, i.e., after a particular activity, the requirements manager is asked whether an additional iteration is required. If affirmative, then the (pending) schedule is automatically readjusted to incorporate the iteration.

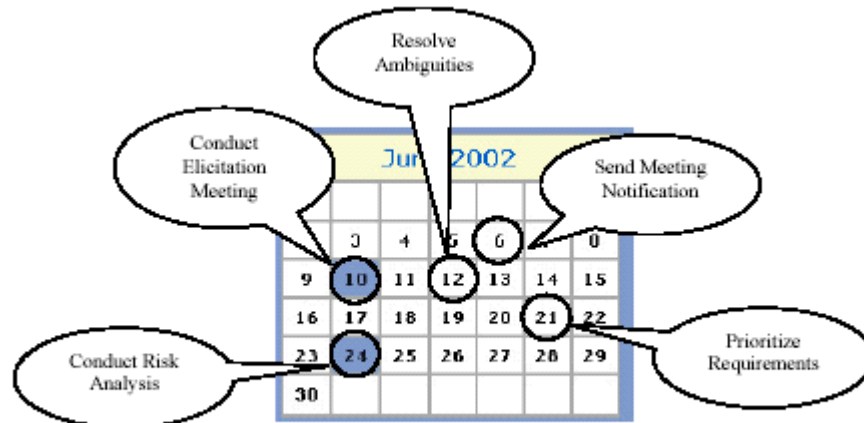


Figure A.4: Scheduler Front-End
Callouts provided by mouse-over effects

The Scheduler enforces structure on the requirements generation process and also assists in performing the development process. It must be noted that the Scheduler is customizable, accounting for missed meetings or passed deadlines.

Email Manager

The Email Manager supports communication among stakeholder. It is invoked after the selection of the participants is made (using the Hierarchy Generator). The Email Manager provides a message template for each selected participant. An email notification is automatically sent to all intended recipients. A schematic representation is shown in Figure A.5.

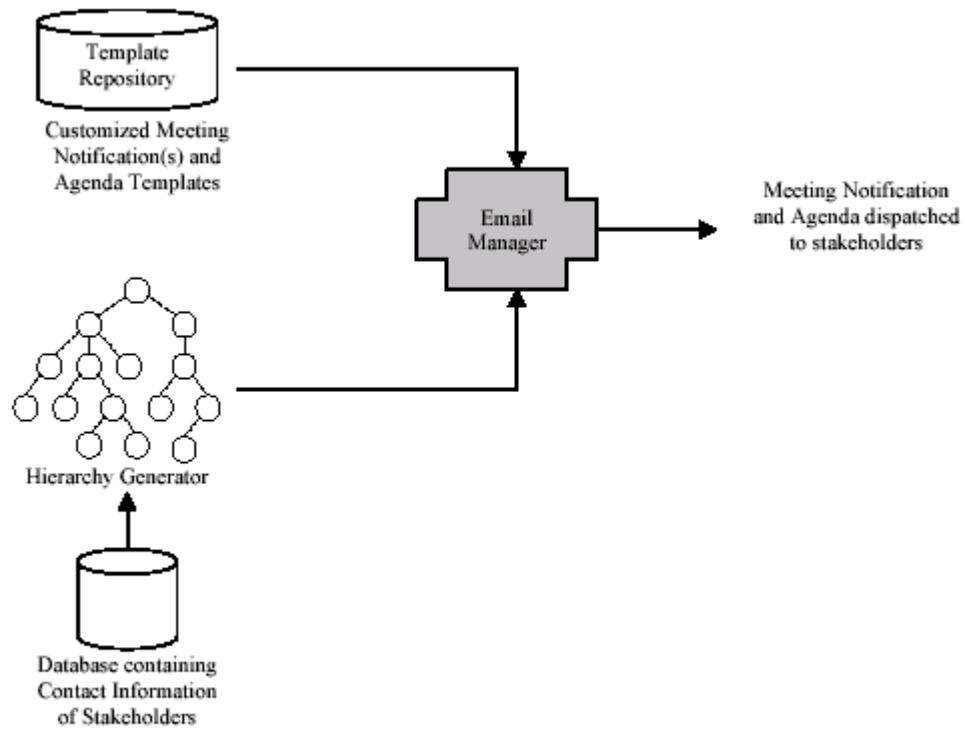


Figure A.5: A Schematic Representation of the Email Manager

The Email Manager also facilitates other communication between the requirements engineer and the customer/users.

4 *it*SRGM Templates

The interactive tool for Requirements Generation provides built-in templates for standardizing the presentation of the information collected during the activities. The following templates are depicted next:

- a. Problem Description Document**
- b. Stakeholder Profiles**
- c. Problem Statement**
- d. Meeting Agenda**
- e. Software Requirements Specification**
- f. Requirements Verification Checklist**

Problem Description Document

1. Introduction

- a. Proposed System overview
- b. Terminology
- c. Organization of Document

2. Business Need

- a. Business Problem (in context to the proposed system)
- b. Business Opportunities (in context to the proposed system)
- c. Existing System Description
- d. Organizational Constraints
- e. Types of Users

3. System Justification

- a. Business Benefits
- b. Shortcomings of Current System

4. System Concept

- a. Proof of Concepts
- b. High Level Functional Requirements

5. Operational Scenarios

- a. Intended Usage Description
- b. Intended User Interactions

6. Business Impact

- a. Proposed system impact on organization
- b. Changes in current organizational setup

7. Rationale

- a. Reasons and convictions for change
- b. Possible risks involved with deployment

8. Conclusions/Recommendations

- a. Development Team recommendations
- b. Go/No Go Decision and justification

Stakeholder Profiles
Stakeholder Name:
Designation:
Department:
<i>Hierarchy Position:</i>
Contact Information:
• Postal Address • Email Address • Telephone (Work): • Telephone (Home):

In order to identify stakeholders, the following information must be collected:

1. Who are the users of the system?
2. Who is the customer for the system?
3. Who else will be affected by the outputs the system produces?
4. Who will evaluate and approve the system when it is developed and deployed?
5. Are there any other internal or external users of the system whose needs must be addressed?
6. Who will maintain the new system?
7. Is there anyone else

--Adapted from [Leffingwell 2000]

Problem Statement for <Project Name>	
The Problem of	<Problem name>
affects	<List of pertinent stakeholders>
the result of which	<Current state of affairs, the cause motivating the need for the change>
benefits of	<Benefit 1>
	<Benefit 2>
	
	<Benefit n>

Sample Meeting Agenda		
Time	Agenda Item	Description
8:00 – 8:30	Introduction	Agenda, Facilities, and rules
8:30 – 10:00	Context	Project status, market needs, results of user interviews, etc.
10:00 – 12:00	Brainstorming	Brainstorms features and requirements for system
12:00 – 1:00	Lunch	
1:00 – 2:00	Brainstorming	Continues
2:00 – 3:00	Feature Definition	Write out 2- or 3-sentence definition for features
3:00 – 4:00	Idea Reduction, Prioritization and Conflict Resolution	Prioritize features
4:00 – 5:00	Wrap-up	Summary

Software Requirements Specification

1. Introduction

- 1.1 Purpose
- 1.2 Document Conventions
- 1.3 Intended Audience and Reading Suggestions
- 1.4 Product Scope
- 1.5 References

2. Overall Description

- 2.1 Product Perspective
- 2.2 Product Functions
- 2.3 User Classes and Characteristics
- 2.4 Operating Environment
- 2.5 Design and Implementation Constraints
- 2.6 Assumptions and Dependencies

3. External Interface Requirements

- 3.1 User Interfaces
- 3.2 Hardware Interfaces
- 3.3 Software Interfaces
- 3.4 Communications Interface

4. System Features

- 4.x System Feature X
 - 4.x.1 Description and Priority
 - 4.x.2 Stimulus/Response Sequences
 - 4.x.3 Functional Requirement

5. Other Nonfunctional Requirements

- 5.1 Performance Requirements
- 5.2 Safety Requirements
- 5.3 Security Requirements
- 5.4 Software Quality Attributes
- 5.5 Business Rules
- 5.6 User Documentation

6. Other Requirements

Appendix A: Glossary

Appendix B: To-Be-Determined List

Requirements Verification Checklist

ORGANIZATION AND COMPLETENESS

- Are all internal cross-references to other requirements correct?
- Are all requirements written at a consistent and appropriate level of detail?
- Do the requirements provide an adequate basis for design?
- Is the implementation priority of each requirement included?
- Are all external hardware, software, and communication interfaces defined?
- Does the SRS include all of the known customer or system needs?
- Is any necessary information missing from a requirement? If so, identify it as TBD.

CORRECTNESS

- Do any requirements conflict with or duplicate other requirements?
- Is each requirement written in a clear, concise, and unambiguous language?
- Is each requirement verifiable by testing, demonstration, review or analysis?
- Is each requirement in scope for the project?
- Is each requirement free from content and grammatical errors?
- Can all of the requirements be implemented within known constraints?

QUALITY ATTRIBUTES

- Are all performance objectives properly specified?
- Are all security and safety considerations properly specified?
- Are other pertinent quality attribute goals explicitly documented?

TRACEABILITY

- Is each requirement uniquely and correctly identified?
- Is each software requirement traced to a higher-level requirement?

SPECIAL ISSUES

- Are all requirements actually requirements, not design or implementation solutions?
- Have internationalization issues been adequately addressed?

Appendix B: Requirements Management Tools A Survey

B.1. Introduction

Requirements Management Tools can assist organizations in defining and documenting requirements by allowing them to store requirements in a central location. Project teams can then access the requirements to determine what is to be developed, and customers can access the requirements to ensure that their needs are correctly specified. Customers and team members (developers) can then participate in informed discussions to seek further clarifications in case of uncertainties. This also aids the process of classifying and prioritizing requirements.

In this appendix, we present a survey of six Requirements Management Tools, highlighting the salient features of each, and describing them on the basis of common, desirable attributes. Table B.1 provides a qualitative comparison of the tools. Table B.2 compares the tools on the basis to cost, performance constraints and platform requirements. Finally, we conclude with a discussion of features in tools that aid the requirements generation process. As a part of this discussion, Table B.3 provides a checklist for identifying the correct Requirements Management Tool for an organization. These tables have been published previously in [Sud 2003].

B.2. Qualitative Comparison of Requirements Management Tools

The tables overleaf provide a qualitative comparison of six requirements generation tools. Features for each have been documented and compared in Table B.1. Table B.2 enlists organizational factors that may affect the selection of a tool.

Requirements Management Tool	Requirements Traceability	Requirements Analysis	Security and Accessibility	Portability and Backend Compatibility	Configuration Management	Communication/ Collaboration	Change Management	Online Publishing	Usability	Requirements Specification/ Documentation
Rational Suite AnalystStudio (Use Case Modeling)	Use Cases are linked to requirements. Any change made to a use case is reflected in the requirements as well. Also supports generation of test cases, facilitated by the integrated suite	Requirements organized into packages stored in a database. From different views of the database, attributes of priority, status, difficulty and version number can be assigned	Requirements protection facilitated by providing attributes, links and user permission privileges	Compatible with MS Word providing context and formatting through a familiar interface	'Requirements Revision History' automatically keeps track of all changes to requirements and records the rationale behind any revision	Stores requirements in central locations enabling access and review by the entire development team. Web interface facilitates viewing and modification of requirements	Any change made to an element is reflected in all other elements automatically. Establishes traceability relations among requirements thereby enabling change assessment	Rational ClearQuest generates automatic change request forms on the Web to manage input from stakeholders	Integration with MS Word provides users with a familiar interface to document and edit requirements. Built-in templates provide quick means to create new projects	Requirements are elicited from use cases providing a structured approach to represent user-system interaction
RDT 3.0	'Requirements Allocation Matrices' forms traceability links between requirements, design, development and test components	Customizable user-defined attributes may be associated to every requirement	Workgroup privileges assign "Add, Edit, View and Delete access to users	Integration with MS Office products	Revision tracking and baseline allocation	Check-Out / Check-in enables the sharing of data between different sites, and the ability to collate this data back to the master database	Change Proposal Management enables a change proposal to be identified, and any data which will be added, changed, or deleted as a result of it being accepted	Shared files are stored on a server. RDT supports up to 255 concurrent users	Data Viewing windows enable concurrent views of different data. Document Views allow data to be edited in tabular format. These views are fully user defined	Automatic requirements capture and syntax parsing directly from existing documents
RTM Workshop 5.0	Defines traceability relationships between individual requirements and requirements and other system elements	Defines different attributes for different types of requirements and set attribute values for individual requirements	Users are categorized in groups and assigned access privileges	Built upon the Oracle database technology	Records project baselines before and after changes are incorporated	RTM e-mails all stakeholders on the status of all proposed changes	Maintains a complete change history including versions of impacted requirements and rationale for each change	Formulates a web interface for database query, discussion threads and updating requirements attributes	Uses word-processing packages from Adobe, Interleaf and Microsoft for its user interface, providing a familiar interface	Parses a document to load requirements into a database

Contd...

Requirements Management Tool	Requirements Traceability	Requirements Analysis	Security and Accessibility	Portability and Backend Compatibility	Configuration Management	Communication/ Collaboration	Change Management	Online Publishing	Usability	Requirements Specification/ Documentation
Telelogic DOORS	Provides 'user-defined, multi-level traceability for relationships' – requirements to test, requirements to design, design to code, etc.	A set of states are associated with each requirement (New, In Review, Partly approved, Approved, More information required, On Hold, Rejected and Applied)	No listed security support. Permits customization of interfaces for every user	Integrates with MS Word and Excel, as well as with other Telelogic tools and third-party applications	DOORS Database Managers manage project baselines, archiving and deleting baselines as the project progresses	On-line Change Proposals and Review Systems, Discussion Threads and distributed data management all collaboration with remote team members	Records the change history of requirements. Notifies team members of changes by email automatically	Telelogic DOORSnet? generates web interfaces to facilitate searching, sorting and editing requirements and submitting change proposals	Provides MS Windows Explorer-like view for project navigation. Customizable user interface by editing colors and graphics to highlight pertinent information	Telelogic DocExpress provides 'one-click' operations to generate requirements for complex systems
Omni Vista OnYourMark Pro	Adds links and tables for each requirement. These tables can have customizable attributes enabling the user to customize traceability links	Requirements analyzed from multiple perspectives – Market, Financial, Development	No listed security support	Compatible with third party application (not listed)	Baselines requirements by ensuring all stakeholders 'sign off' on the latest agreement	Online web publishing facilitates execution of customized queries and sharing of information, all through the web interface	Requirements Triage permits evaluation of impact on overall project based on individual factors	A snapshot of the requirements can be published to the web or exported to an XML format	Detailed graphical representations indicating the effects of adding features on resources (budget, time, schedule, etc.)	Tabular representation recording, tracking and annotating requirements. Permits importing requirements from third party tools
Starbase Caliber-RM	Multiple method of traceability visualization enable users to understand effect of impact of requirements change on other requirements, tests, tasks and/or source code	Spreadsheet views permit sorting and prioritizing requirements according to cost and value. Different views may be created based on the data collected	Centralized repository provides security, visibility and availability to all requirements data	'Open Architecture' permits requirements to be linked to external source code generators, test suites and project management applications	Independently versions each requirement. Baselines specific versions providing users the option to compare requirements versions to assess volatility	Online discussions capture information more effectively than asynchronous (email) communication. Immediate notification of changes sent to all stakeholders	Each change creates a unique history record; differences between one version of a requirement and another are highlighted with the reason for the change recorded.	Provides team members with a network connection access to requirements data, imposing minimal bandwidth requirement	GUI is customizable using wizards. Complex scripting languages not required to draft requirements. Can be customized to support many requirements processes	Documents requirements and glossaries to define industry terms, project terms, corporate terms, etc.

Table B.1: A Qualitative Comparison of Requirements Management Tools

B.3. Organization Factors and Constraints of Requirements Management Tools

Requirements Management Tool	Cost (in U.S. dollars)	Operating Expenses/ Licensing Fees	Platform Requirements			
			Processor	Disk Space (Client + Server)	Operating System	Other Requirement(s)
Rational Suite AnalystStudio (Rational Requisite Pro Component)	\$ 2034 per user	Includes 1 year of support	>= 300 MHz 128 MB RAM recommended	96 MB	Microsoft Windows NT 4.0, or Microsoft Windows 98, 2000 or Millennium Edition	Word Processor: Microsoft Word 97, 2000 or 2002 Database: Microsoft SQL Server 7.0 or greater, Oracle 7 or 8 (creates MS Access databases by default) License: Licenses can be node locked or floating. A license manager (flexLM from GLOBETrotter) is provided with AnalystStudio.
RDT 3.0	\$ 2995 per user for software	\$ 650 per user support for 1 year	>=200 MHz >= 32 MB RAM recommended	60 MB	Microsoft Windows 95/98/NT 4.0/2000	Word Processor: Microsoft Word 97, 2000 or 2002
RTM Workshop 5.0	N/A	N/A	>= 200 MHz >= 64 MB RAM recommended	250 MB	Microsoft Windows 95/98/NT 4.0 and UNIX	Word Processor: Microsoft Word or Adobe FrameMaker Database: Oracle 7 or 8 License: Uses the floating networking license manager
Telelogic DOORS	N/A	N/A	>= 200 MHz Follow Manufacturer recommended RAM	40 MB	Microsoft Windows 98/2000/NT	License: Supports network (floating, node locked, etc.) licensing
Omni Vista OnYourMark Pro	N/A	N/A	N/A	N/A	Microsoft Windows 98/2000/NT	N/A
Starbase Caliber-RM	N/A	N/A	>= 200 MHz >= 256 MB RAM recommended	125 MB	Microsoft Windows 98/2000/NT Java-enabled web client	Database: Caliber-RM ships with a server version of the object-oriented database (OODB), Versant Object Repository, which is also commercially available License: Caliber-RM may be licensed either by seat or on a concurrent user basis

Table B.2: Organizational Factors

B.4. Discussions and Conclusions

There are numerous tools¹³ available that support the requirements management process. This report surveys six popular tools. Some tools have more features that are more important than others. The leading products have features that make the process easier, efficient and faster for the administration and the team members.

We have enumerated some features that are crucial in a requirements management tool. We have also provided a table that provides questions to be answered while selecting a requirements management tool.

Ability to manage versions and changes

At some stage of development, a project should define a requirements baseline, that is, a specific collection of requirements that a particular release will contain. A requirements management tool must record a history of the changes made to every requirement. This helps capture previous decisions and rationales.

Store requirements attributes

Comprehensive information and attributes about requirements should be stored to promote understandability, for customers as well as developers. Requirements must be understandable by all concerned stakeholders. Selected individuals should be able to update those attributes. Requirements management tools automate the generation of several system-defined attributes, such as date created and version number, and allow the definition of additional attributes of various data types. Attributes such as author, person responsible, origin, rationale, release number, status, priority, cost, difficulty, stability and risk further augment understandability.

Link requirements to other system elements

Tracing individual requirements to other system components helps ensure that developers do not inadvertently overlook any requirements during implementation.

¹³ A fairly comprehensive listing of all available requirements management tools can be obtained at <http://www.incose.org/tools/tooltax.html>

When analyzing the impact of a change proposed in a specific requirement, the traceability links reveal the other system elements that the change might affect.

Track Status

A good requirements management tool helps track the status of each requirement during development, and thereby supports overall project status tracking. If a project manager knows that 55% of the requirements allocated to the next release have been implemented and verified, 28% are implemented but not verified, and 17% are not yet fully implemented, then he or she has good insight into project status.

View requirement subsets

Requirements Management tools can help sort, filter, or query database(s) of requirements to view subsets of the requirements that have specific attribute values.

Control access

Access permissions should be set for individuals or groups of users. This prevents miscreants and inadvertent accesses to requirements.

Communicate with stakeholders

Most requirements management tools let team members discuss requirements issues electronically. E-mail messages can notify the affected individuals when a new discussion entry is made or a requirement is changed.

Backend Compatibility with Editors

Most requirements management tools described in this section are compatible with popular editors, such as Microsoft Word. This is helpful to distribute and edit requirements and interchange information with other tools.

Publishable Web Interfaces

Recent tools have the ability to publish information on the Internet with a click of a button. This helps sharing information and promotes remote collaboration, particularly in development efforts requiring remote collaboration

No.	Question	Answer
1	Does the tool support analysis of existing documentation?	Y / N
2	Does the tool support automatic parsing of documents?	Y / N
3	Can the tool compare different versions of the same document (detect changes)?	Y / N
4	Does the tool support the reading of requirements from external tools?	Y / N
5	Does the tool support requirements traceability?	Y / N
6	Does the tool have the capability to categorize requirements as they are elicited and documented?	Y / N
7	Can the tool capture or generate graphical representations (system architecture, analysis graphs, tables, etc.) of the system?	Y / N
8	Can the tool link requirements to other system elements (use cases, design, code, test cases, etc.)?	Y / N
9	Does the tool support the association of attributes to every requirement?	Y / N
10	Can the tool detect inconsistencies in requirements (incomplete traceability links, missing attributes, etc.)?	Y / N
11	Does the tool provide information of requirements verification (whether a requirements was implemented or not, who did it and when)?	Y / N
12	Does the tool maintain a history of requirements changes?	Y / N
13	Does the tool have the ability to baseline requirements document and then compare different baseline?	Y / N
14	Does the tool provide security to prevent inadvertent or malicious modifications to information?	Y / N
15	Does the tool support the generation of the Software Requirements Specification (SRS) in standardized formats?	Y / N
16	Does the tool check for inconsistencies (spelling errors, etc.) in the SRS?	Y / N
17	Does the tool support querying?	Y / N
18	Does the tool integrate with other CASE tools?	Y / N
19	Does the tool provide publishable web interfaces?	Y / N
20	Does the tool support communication among team members?	Y / N
21	Is support offered for single and multiple concurrent users?	
22	Which platforms and operating systems does the tool run on?	
23	Does the tool use a commercial or a proprietary database to store requirements?	Y / N
24	What are the hardware and software requirements for the tool?	
25	Does the tool provide an interface for manipulating information?	Y / N
26	Does the tool manage change by adjusting for a change in all pertinent places?	Y / N
27	Does the tool support web access to the information stored in the database?	Y / N
28	Does the tool have a warrantee?	Y / N
29	Are user manuals online or hard copies are provided?	Y / N
30	What kind of support does the vendor provide?	

Table B.3: Checklist for selecting a Requirements Management Tool

Existing Requirements Management Tools

Rational Suite AnalystStudio – <http://www.rational.com/products/astudio/index.jsp>

RDT 3.0 (Requirements Design and Traceability) – <http://www.igatech.com/rdt>

RTM Workshop 5.0 –

http://www.chipware.com/sub1.php3?section=rtm&sub=RTM_Overview

Telelogic DOORS – <http://www.telelogic.com/products/doorsers/index.cfm>

Omni Vista OnYourMark Pro – <http://web.uccs.edu/adavis/UCCS/index.htm>

Starbase Caliber-RM – <http://www.borland.com>

Wiegiers K.E. – Automating Requirements Management, <http://www.processimpact.com>

Appendix C: Requirements Specification Guidelines

The final product of the requirements generation process is a documented agreement between the customers and the development group about the product to be built. This agreement is referred to as the Software Requirements Specification, or the SRS.

Appendix A provides a template for creating the SRS. In what follows, we describe each section indicated in the template. The information represented herein and in the template is adapted from [Weigers 1999].

. The intent of providing this detailed information is to provide the requirements engineer with a resource outlining the information desired from the customer. Efforts may be oriented accordingly.

1. Introduction

The introduction presents an overview of the SRS to help the reader understand how the document is organized and how to read and interpret it.

1.1 Purpose

Identify the product/system whose software requirements are specified in this document. If the SRS pertains to only a part of a larger system, identify the portion or subsystem that it addresses.

1.2 Document Conventions

Describe any standards or typographical conventions that are followed when writing this SRS, including text styles and notations.

1.3 Intended Audience and Reading Suggestions

List the different readers to whom the SRS is directed, *e.g.* developers, project managers, marketing staff, users, testers, and documentation writers.

1.4 Product Scope

Provide a short description of the software being specified and its purpose, including benefits and goals. Relate the software to corporate goals or business strategies, if any.

1.5 References

List any documents or other resources to which this SRS refers. These might include standards, contracts, high-level system requirements specification, con-ops document, etc.

2. Overall Description

This section presents a high level overview of the product being specified, the environment in which it will be used, the anticipated users of the system, and the known constraints, assumptions and dependencies. Information can be elicited from the Context Diagram and Constraints Document produced during the Problem Synthesis process (Section 3.2.2.2.3).

2.1 Product Perspective

Describe the context and origin of the system being specified in this SRS. State whether this system is a part of a larger system, the next release of an already existing application, a replacement for existing system or a new, self-contained product.

2.2 Product Functions

Summarize the major functions the system must perform. Details are provided in Section 4 of the SRS, so only a high-level summary needs to be specified. A high level use-case model or a class diagram can also be included in this section.

2.3 User Classes and Characteristics

Identify the user classes that you anticipate will use the system and describe their characteristics and expectations. Some requirements may be specific to certain user classes. Provide the means for representing this differentiation.

2.4 Operating Environment

Describe the environment in which the software will operate, including the hardware platform, operating systems and versions, and other software components or applications with which it must peacefully coexist.

2.5 Design and Implementation Constraints

Identify any issues that can restrict the options available to the developers and describe why they are constraints. Constraints can include the following:

- Specific technologies, tools, programming languages, and databases that must be used or avoided
- Required development conventions or standards
- Corporate policies, government regulations, or industry standards
- Hardware limitations, such as system timing, memory restrictions, etc.

2.6 Assumptions and Dependencies

List any assumed factors that can affect the requirements stated in the SRS. These factors, if realized, may inadvertently affect the overall development effort. Such factors include: changes in market demand (obsolescence) and technological improvisation. Also identify any dependencies the system has on external factors, especially if the system is a part of a larger development effort.

3. External Interface Requirements

Use this section in the SRS to specify any requirements that help ensure the new product connects properly to external components. The context diagram shows that external interfaces at a high level of abstraction. One also places detailed descriptions of the data and control components of the interfaces in this section.

3.1 User Interfaces

State the software components for which a user interface is needed. Describe the logical characteristics of each user interface. The following are some characteristics that can be included:

- GUI standards or style guides that are to be followed
- Screen layout and resolution constraints
- Standard buttons, functions, or navigation links that will appear on every screen
- Shortcut keys
- Error message display standards.

3.2 Hardware Interfaces

Describe the characteristics of each interface between the software and hardware components of the system. This description might include the supported device types, communication protocols, etc.

3.3 Software Interfaces

Describe the connections between the system and other external software components including, databases, operating systems, tools and libraries.

3.4 Communications Interface

Describe the requirements associated with any communication functions the system will use, including e-mail, browser, network communications standards or protocols, electronic forms, etc.

4. System Features

This section enumerates all features that must be implemented in the system.

4.x System Feature X

State the name of the feature in a few words. Subsections 4.x.1 through 4.x.3 will be repeated for each system feature.

4.x.1 Description and Priority

Provide a short description of the feature and indicate whether it is of high, medium or low priority. Other attributes such as risk, effort and cost should also be specified.

4.x.2 Stimulus/Response Sequences

List the sequences of user actions, or signals from external devices, and system responses that define the behaviors for this feature.

4.x.3 Functional Requirement

Itemize the detailed functional requirements associated with this feature. These are software capabilities that must be present for the user to carry out the services provided by the feature or to perform the task for which the system is built.

5. Other Nonfunctional Requirements

Use this section to list any nonfunctional requirements other than external interface requirements and constraints.

5.1 Performance Requirements

State any performance requirements for various usage scenarios, and explain their rationale to help the developers make suitable design choices. Requirements for memory, disk space and processor speed may also be specified in this section.

5.2 Safety Requirements

Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be incorporated in the system to prevent potentially dangerous situations. Identify any safety certifications, policies or regulations to which the product must conform.

5.3 Security Requirements

Specify any requirements regarding security, integrity, or privacy issues that affect the use of the product and protection of the data used or created by the product. Define any user authentication or authorization requirements.

5.4 Software Quality Attributes

Specify any additional product quality characteristics, such as those specified in figure 3.13. These characteristics should be specific, quantitative and verifiable when possible.

5.5 Business Rules

List any operating principles about the system, such as which individuals or roles can perform which functions under which specific circumstances.

5.6 User Documentation

List the user documentation components that will be delivered along with the software, such as user manuals, online help, and tutorials. Identify any known user documentation delivery formats or standards.

6. Other Requirements

Define any other requirements that are not covered elsewhere in the SRS, such as internationalization requirements or legal requirements. This section may be omitted in the event that no such requirements exist.

Appendix A: Glossary

Define all the terms necessary for the reader to properly interpret the SRS, including acronyms and abbreviations.

Appendix B: To-Be-Determined List

Compile a numbered list of the TBD (to be determined) reference that remain in the SRS so that they can be tracked to closure.

Vita

Rajat Sud was born on August 13th, 1979 in Calcutta, India. Having lived in Bombay since a very early age, he graduated from the University of Bombay, in June 2001, with a Bachelors' degree in Computer Engineering. Subsequently, he has been pursuing a Masters' degree in Computer Science at Virginia Tech, Blacksburg, Virginia.

During his Bachelors' degree, he was awarded the 13th rank in the dean's list for his performance in the under-graduate curriculum. He has also been a member of technical organizations, and has been the chairperson for an all-India collegiate event held in the spring of 2000.

During his Masters' degree, he has served as a Graduate Teaching Assistant, and has been conducting research in the field of Requirements Engineering. Rajat is also a member of ACM.

On completion of his degree, Rajat will be working with an IT consulting company, based in Princeton, New Jersey. He can be reached at rsud@vt.edu.

Rajat Sud