**Capturing Requirements Meeting Customer Intent:**
**A Structured Methodological Approach**

Markus K. Gröner

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

Doctor of Philosophy
in
Computer Science and Applications

Approved:

_____
James D. Arthur, Chair

_____ _____
J.A.N. Lee Richard E. Nance

_____ _____
Linda H. Rosenberg MaryBeth Rosson

May, 2002
Blacksburg, Virginia

Keywords: Requirements Engineering, Software Engineering, Methodology

**Capturing Requirements Meeting Customer Intent:**
**A Structured Methodological Approach**

Markus K. Gröner

**(ABSTRACT)**

Product quality is directly related to how well that product meets the customer's needs and intents. It is paramount, therefore, to capture customer requirements correctly and succinctly. Unfortunately, most development models tend to avoid, or only vaguely define the process by which requirements are generated. Other models rely on formalistic characterizations that require specialized training to understand. To address such drawbacks we introduce the Requirements Generation Model (RGM) that (a) decomposes the conventional "requirements analysis" phase into sub-phases which focus and refine requirements generation activities, (b) constrains and structures those activities, and (c) incorporates a monitoring methodology to assist in detecting and resolving deviations from process activities defined by the RGM. We present an empirical study of the RGM in an industrial setting, and results derived from this study that substantiate the effectiveness of the RGM in producing a better set of requirements.

**ACKNOWLEDGEMENTS**

I am forever thankful for the wisdom and guidance my advisor Dr. James D. Arthur has provided me over the past seven years. He and I have spent an incredible number of hours together in defining and refining the necessary components of this dissertation. Thanks to his continuous support and pushing I finally completed this task.

My deepest love and thanks go to my wife Miriam. Her love and support over the long years made it possible for me to finish this work.

Last but not least I would like to thank my parents, who, without questioning have always provided me the necessary support so that I was able to acquire the education I wanted.

## TABLE OF CONTENTS

**LIST OF TABLES**

**LIST OF FIGURES**

# Chapter 1. Introduction

## 1 INTRODUCTION

This dissertation presents a Requirements Generation Model (RGM) designed to enhance the quality of established requirements for software development. This chapter motivates the need for continued research of current approaches in requirements generation, and the reasons for creating a Requirements Generation Model.

### 1.1 Background (Area of work)

The work presented here falls within the area of requirements engineering. The information shows a progression from early software development through current research in requirements engineering, and identifies an insufficiency thereof which can be satisfied by the proposed RGM.

#### 1.1.1 Chaos

An understanding of the current deficiencies in the software development process can be gained from a study of the evolution of the software development industry. The first software developers could not foresee the future potential of their emerging industry and therefore did not set in place any farsighted processes to govern the growth. At that time, cost of computer systems was extremely high, and availability of systems sparse. The systems were programmed by only those intimately familiar with the systems' design itself, therefore was no need for good development processes.

Early software development consisted of generating code that programmed machines to perform repetitive calculations faster than could be done by a human [40]. The functions were straightforward and the end goals specific from the start of the development, so a great deal of pre-programming analysis was not necessary. Code was specific to the machine and, due to storage constraints, written in a precise, compact manner. However, advances in storage that made it less expensive and available in large capacities relaxed the need for "precise" code. And as software became increasingly more sophisticated – and complicated – high-level programming languages were developed. Programming language elements such as "goto", "jump", and "subfunction" gave developers flexibility

to create and alter code as needed. The focus was on the end product, and developers could modify the code until it finally ran as required; forgotten or new requirements could easily be added by inserting a “goto” to execute newly added functionality. While facilitating the modification process, this flexibility was available without any structure or governing guidelines, and the practice of inserting quick fixes became rampant. The result of programming without structure was chaos, or more often referred to as “spaghetti code.” Maintenance became a nightmare as the lack of coding structure and documentation made it close to impossible to find coding errors [30]. Quick fixes created new problems that could not be foreseen.

### 1.1.1 Software Development Lifecycle

Recognizing the problems developers faced due to the lack of structure in development led the software development community to utilize more rigorous approaches. The Software Development Lifecycle (SDLC) introduced five phases: Requirements Analysis, Design, Implementation, Integration and Testing (I&T), and Maintenance [34]. These phases are found in software development models such as the Waterfall [38] and the Spiral [6], and in theory address all aspects of efficient and reliable software production. Although logic seems to dictate that diligent attention should be directed towards the first of these five phases, Requirements Analysis, to establish a firm and reliable support for the remaining phases, most of the emphasis by software developers continued to be placed on the end product. So, although the advent of the SDLC was a forward move by the industry, attention was focused on the third and fourth phases, the Implementation and I&T phases, which prompted the development of automated test tools designed to detect deeper coding errors. Although those tools are very useful, one must wonder if they would be quite so necessary if the requirements were accurate. Furthermore, studies show that 60-70% of time attributed to the SDLC is actually spent on Maintenance, the fifth phase [13].

Maintenance includes fixing errors, changes to code based on wrong requirements, or additions (also called enhancements) due to previously unknown requirements – all of which could theoretically be avoided with adequate attention to the initial phase of deve l-

opment. Later, the advent of high-level languages allowing for modular code (through procedures and functions) enhanced the readability, testability, and maintainability of code [39]. Yet, this advance may have also contributed to a "fix it later" attitude, which again focuses on the latter development stages rather than establishing a strong foundation up front.

### 1.1.2 Shifting focus in the SDLC

Modular code made fixing errors easier, but this maintenance still incurred an added cost to the overall development. The high cost associated with system/software maintenance prompted software developers and researchers to shift their focus within the SDLC. Instead of concentrating on the Implementation and I&T phases, which ensure good coding and validation through testing, the focus shifted to the Design phase – second in the SDLC. Recognizing that "step-wise" refinement [32], which involves reducing large tasks into small, easier to handle units and iterating until all units were solved could be successfully applied to design, the Design phase was separated into high-level and low-level phases [33]. These two "sub-phases" each had their own sets of objectives and resulting artifacts. The high-level design phase yields a big picture of the development project. The low-level design phase allows for detailing and refinement of each separate component mentioned in the high-level design phase [36]. This move showed progress in the industry, as it demonstrated a recognition by developers that some problems requiring maintenance at the end of the process could be avoided with more care applied at the beginning, saving both time and money. The Design sub-phases created openings for new processes intended to improve the quality of the design itself.

### 1.1.3 Design processes

The new processes contributed to the structure of the two design phases. Among these methods, "structured analysis" and "structured design" focus on activities to create better design artifacts from requirements [4]. Others, especially the Fagan Inspection Process [18], focus on verifying the qualities within the design documents. This is done through rigorous, formal inspections of the high-level, as well as the low-level design documents. All of these new measures improved the overall quality of the development.

*Remaining problems*

The improvement in I&T (through the use of automated tests), the improvement in coding through structure and high-level programming languages, and the division of the Design phase into two "sub-phases" led to enhanced software products. However, several problems still plague the software development community. For example:

- Scope creep causes requirements to change constantly
- The cost of fixing post-development errors continues to increase
- Wrong products (not used) are being developed at substantial cost
- Questionable quality exists in products with high risk of failure
- Amorphous requirements continue to exist

## 1.2 Motivation for Requirements Generation Model

The progression from chaos to a process supporting a formal approach to Requirements Engineering has been a long and difficult path. This path has led the software community through several stages: "ad hoc coding," "the art of coding," "the science of coding," and finally to the realization of necessary repeatable processes through engineering [26]. Requirements engineering as a discipline stems from the overall software engineering realm. The goal of requirements engineering research is to provide the software development community with methods, processes, and tools that, when correctly applied, result in a quality software product that meets requirements established by the customer. Two general types of approaches emerged through this research, one focusing on customer involvement and a second one stressing the creation of more formalized approaches to requirements generation.

### 1.2.1 Customer-involved approaches

The intention of not just including, but involving the customer in the requirements generation process is to receive early, valuable feedback from the customer. Two approaches that emphasize customer involvement are Participatory Design (PD) [29] and Joint Application Design (JAD) [34]. In PD, the customer works closely with the requirements engineer throughout the SDLC until final product deployment. The goal of PD is to narrow the "knowledge gap" between the customer and requirements engineer,

and to allow the customer early familiarization with the new product, thereby minimizing the learning curve once the product is delivered. PD requires the software engineer to work closely with the customer, learning the customer's job (work currently done either manually or with an existing system), expending many hours in shadowing, one-on-one interviews, and actually performing the customer's task before determining the actual new necessary requirements. A disadvantage to PD is the high initial cost due to the time added to development as both parties become acquainted with each other's tasks. Few software development companies use PD because of this high cost that many customers are not willing to pay. Additionally, political issues may arise as the software engineer becomes closely involved with the customer.

JAD is becoming much more popular within software development companies due to its ability to gather a great deal of information from many people in a very short amount of time. Contrasting to PD, JAD is based on an initial requirements-gathering session, involving both the customer and developers, aimed at harvesting the greatest amount of information from all concerned parties. However, a good facilitator is necessary to make JAD sessions successful. Certain problems must be avoided, such as "put-downs," "pulling rank," and other behaviors that adversely impact the brainstorming during a JAD session. A greater 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 and to establish the correct set of requirements.

Both PD and JAD are good approaches to involve the customer in the requirements identification process. They are, however, only tools and do not provide the necessary structure to gather all necessary requirements.

### 1.2.2 Formal Approaches

The advocates of formal approaches to requirements generation base their reasoning on the mathematical grounds of programming languages and computers. The theory is that a correct system can be established if the requirements are represented in a formal language or system, allowing those requirements to be proven correct. Requirements Definition

Languages (RDLs) such as Software Requirements Engineering Methodology (SREM) [1],and Program Definition Language (PDL) [40] aim to overcome, through formalization, the challenges of natural language approaches, such as requirement ambiguity, incompleteness, readability, and understanding. Further, several of the RDLs are formal enough that they can be used for automated design and coding. There is no question that formal approaches offer a significant benefit; nor is there a question that formal approaches, when applied correctly, can result in a correct system. There is, however, one paramount question that remains: *Does this correct system do what the customer needs it to do?* If the answer is no, what good is the correct system? While system correctness is certainly important, there are many disadvantages to formal approaches. Several key issues are listed below:

- Most requirements engineers are neither familiar with RDLs, nor can they apply them
- RDLs cannot be understood or read by most customers
- RDLs widen the knowledge gap
- RDLs prove the software meets the requirements; they do not ensure the requirements meet customer need

### 1.2.3 Localized approaches without a guiding structure

Both the formalized and customer-involved approaches consist of methods aimed toward a local goal - 'local' in the sense that their entire focus is grounded in a single goal as described below:

- PD – close working relationship between requirements engineer and customer
- JAD – facilitation of meetings to better extract requirements
- RDLs – formalization of requirements to prove correctness

Each method is not without merit, and combining one or more of these methods will improve the difficult task of generating customer requirements. However, each method lacks a formally defined structure within which it should be applied.

Without this structure, it is difficult to put the various methods into perspective, and even more difficult to bind them through a common set of objectives. If we take a step back-

wards and look at the phases of the Waterfall model, for example, we would expect to see that existing methods defined within the requirements phase would adequately address the requisite activities. If this were true, we would already have a viable approach to generating high-quality requirements. The fact is, however, that all existing requirements methods have a focus too narrow to truly evoke high-quality requirements. Collectively, they comprise many effective parts of an as yet non-existent whole: an overarching structure to guide the requirements generation process.

### 1.2.4 Ad hoc Requirements Generation

By definition, an *ad hoc* approach is "for the particular end or case in hand without consideration of wider application." As illustrated above, requirements generation approaches abound, each with it's own necessary goal, but none with a view towards the larger picture of the entire requirements generation process. This results in a lack of structure and a lack of guidance throughout the requirements generation phase, eliciting nothing but inconsistent results within the development of each individual system, as well as inconsistency throughout the industry. Neither the customer nor the requirements engineer has control of the process, and the customer has no method by which to compare approaches taken by different development groups. Companies soliciting bids for large, high-risk projects often require development companies to follow IEEE standards [25] and to be ISO9001 certified in order to merit consideration for the award [41]. These standards ensure many quality control methods and are certainly worthwhile, but there is a widespread misconception as to what they actually encompass.

One item notably absent from these standards is the consistent verification of a thorough requirements generation process. Therefore, a customer cannot compare the bids on this most critical phase of the development process. Once the contract has been awarded, managing risk at the requirements generation level becomes difficult at best when the development company has no structure or methods in place to avoid problems or to prevent errors from being introduced during the generation process. The ad hoc nature of requirements generation is evidenced by last-minute meetings with no announced agenda, unclear tasks and responsibilities of meeting participants, inefficient time management.

All of these outcomes are symptoms of an inadequate approach which results in lost time and wasted money. A model providing consistent verification of a thorough requirements generation process can accomplish the significant goals of consistency across the industry, higher quality products, and greater customer satisfaction.

### 1.3 Problem Statement

The contributions of the approaches and methods discussed in the previous section all aim at achieving a thoroughly accurate set of requirements. However, as we have shown, the lack of an overarching structure prevents each of these from ultimately achieving this goal. Furthermore, because of the narrow focus of these approaches, even when a quality set of requirements is generated, customer intent is often not met. Therefore, the ultimate goal of requirements generation should *be capturing and defining requirements as intended by the customer*.

It is apparent that this goal is oftentimes not achieved due to a) projects getting canceled before completion because the requirements have changed and can no longer be met by the system being built, and b) systems delivered but never used because they don't function as intended by the customer.

Statistics show that greater than 50% of all initial requirements change over the lifetime of a system [14]. Requirements change is unavoidable, due to economic, environmental, and other impacts over the development time of the project. However, we would like to conjecture that a high percentage of those changes are not based on these impacts, rather on initial requirements not being generated as intended by the customer. Those "erroneous" requirements necessarily had to change; but, if they are not discovered early in the SDLC, require a high cost for fixing at a later time. If undiscovered through project deployment, the cost, according to Barry Boehm [8], is greater than 400% of what it would have cost if fixed during an early phase of development. Worse, such a system also has a high risk of failure, and in some cases such as military systems, that failure could possibly carry the cost of human life. Therefore, it is imperative that a strict and reliable structure and process be implemented which can ensure that requirements are captured as

intended by the customer during the requirements generation phase, and that a system of proofing those requirements is in place to govern the process across the industry.

### 1.3.1 Key Issues

From initial chaos to current requirements engineering, many advancements have been made in improving not just the SDLC but also the activities within. However, if we compare the problems faced during the early advances in software development to those we are faced with today, we find many commonalities, as well as remaining problems. Some of those problems are reiterated below:

- Constantly changing requirements
- High cost of fixing errors
- Wrong product delivered at high cost
- Slipping schedules
- Questionable product quality

This suggests that the advances made have not comprehensively addressed these problems. But each of these problems can be resolved by adequately addressing the activities and processes within the requirements phase under one overarching umbrella structure.

*Deficiencies of current "approaches"*

This dissertation establishes that the existing approaches to requirements generation are inadequate. The disadvantages are myriad and extend beyond even what has been ascertained here, and they vary from extreme to extreme. These various approaches promote knowledge gaps, technology gaps, domain gaps; they focus too narrowly on single aspects of the SDLC; they are too broad to focus on any aspect effectively; they are applied at random and without any governing methodology which ensures reliable, systematic requirements generation.

A model needs to focus on the goals and objectives of the entire process, including the individual activities and phases within such a process, and provide necessary constraints to provide guidance throughout the process. A process without controls can breakdown, or fail to provide the necessary support needed to accomplish the purpose of the process.

Furthermore, without proper controls in place, a process can deteriorate, which often becomes observable only in the reduced quality of the product generated with or during the process.

Requirements engineering can never be able to achieve its intended purpose without a controlled, comprehensive model for requirements generation.

## 1.4 Problem Solution Approach

To address the issues discussed and to form a comprehensive approach to requirements generation under one overarching structure, we have created the Requirements Generation Model (RGM). The RGM consists of a framework and a monitoring methodology.

### 1.4.1 Framework

The framework of the RGM creates a constraining structure within which the customer and the requirements engineer operate to generate requirements. The framework consists of two major phases: indoctrination and requirements capturing. The requirements capturing phase further breaks down into three sub-phases: preparation, elicitation, and evaluation. The framework is constrained through protocols to which the customer and the requirements engineer must adhere during the requirements generation process. Furthermore, guidelines are included with the framework offering "good practices" which, when followed, can further enhance the requirements generation process and ultimately have a positive impact on the quality of the requirements. This dissertation shows that this framework addresses the following key issues:

- creating a structure and a systematic approach to requirements generation
- narrowing the gaps found between customer and requirements engineer
- focusing on all activities of requirements generation and meaningfully constrain the process
- progressively elaborating and refining requirements as they are generated

### 1.4.2 Monitoring Methodology

The RGM requires a means to continuously control the requirements generation process and to provide solutions when activities take place outside the control limits. For this

purpose we have created a Monitoring Methodology that is applied specifically during the requirements elicitation sub-phase. This monitoring methodology follows standard methods by including both procedures and methods. Here, procedures are a) continuously applied to monitor activities within the requirements elicitation sub-phase and b) indicate methods when irregularities are detected in the requirements elicitation sub-phase. To clarify, an example of a procedure within the monitoring methodology is:

*Recognize when a solution is suggested during requirements elicitation.*

A method to resolve this irregularity is:

*Remove solution suggestions from requirements artifacts if these have been included. Determine impact of solution suggestions on other requirements.*

The application of the procedures and methods of the monitoring methodology by the requirements engineer or customer is discussed in detail in Chapter 4.

### 1.4.3 Strategy

The Requirements Generation Model is designed with several strategic aspects in mind:

- Provide necessary overarching structure and constraints.
- Provide means to keep the process in control.
- Prescribe no specific methods or tools that should be used in the process of requirements generation.
- Make the Requirements Generation Model a solid foundation for requirements generation, but make it flexible and extensible so as to adjust to any development environment.
- Minimize extra time and cost the Requirements Generation Model could introduce. (The added time and cost over an established development effort has been one obstacle preventing other methods from enduring.)

Guided by these strategies, and focusing on improving the key issues reported here, we have created the Requirements Generation Model.

### 1.5 Blueprint of Dissertation

The remaining chapters of this dissertation are set as follows. Chapter 2 discusses the background of this dissertation. We focus on work that has previously been done either in this area or work that parallels the thinking in this dissertation. Chapter 3 presents the details of the RGM framework. Chapter 4 focuses on the monitoring methodology. A pilot study that was conducted using a subset of the RGM is presented in Chapter 5 together with the data analysis and discussion of the pilot study findings. A final summary, concluding remarks, and suggestions for future application in requirements engineering close of the main dissertation in Chapter 6. Following the last chapter are references, appendixes and the author's vita.

# Chapter 2. Background

## 2 BACKGROUND

This chapter presents background information that has had major influence on the research presented in this dissertation. The two sections in this chapter are structured to present 1) aspects of the Software Development Lifecycle (SDLC), and 2) work within the area of Requirements Engineering.

### 2.1 Software Development Lifecycle

The SDLC for a software product spans the time from the initial conception of the idea (before or during concept definition), through five stages of development, to the day the product is taken out of service. As software development has evolved, various models and approaches have been defined to structure and describe the different phases of the SDLC. To enhance the effectiveness of these models and approaches, methods have been developed that describe actions to meet specified objectives as well as aid in improving the artifacts stemming from the various stages of the SDLC. Some of these methods are independent of the models and can be deployed among various models and approaches, whereas others are integral parts of the models and approaches.

In the sections of this chapter we focus on the more influential models and approaches, as well as methods that have impacted the SDLC. We show those aspects of each that have been significant to our research.

#### 2.1.1 Models and Approaches to Software Development

Software Development Models and Approaches define processes for the SDLC. These processes generally consist of well-defined phases, activities, and artifacts to guide the software development effort. Models provide two important facets to software development:

- An abstraction of the SDLC – this gives the software engineer as well as the customer of the product a high-level overview of the project. It allows for the partitioning of the project into smaller, well-defined phases (a structure) that are easily

manageable. Furthermore, important milestones can be set based on these phases with physical deliverables bound to them (important to the customer).

- An identification of objectives – each phase of the model providing structure to the SDLC is defined by its objectives. To reach these objectives, specific activities are described within each phase. Some of these activities require that generation of artifacts (documentation) that is verified or analyzed at the conclusion of each phase to substantiate that the phase objectives have been met. We find that some SDLC models leave the activities within the phases wide open to the software engineer's discretion, others provide a set of guidelines that should be followed, while others are completely restrictive in their activities.

The five models and approaches we describe in this section all span the entire SDLC with well-defined phases and activities. Some of these models and approaches are open enough, though, to be enhanced or supplemented by further activities, such as those found in more independent methods described later in this chapter.

*Waterfall*

The Waterfall model (Figure 2-1) is recognized as the traditional software engineering model. The proposed model by Royce [38] consisted of five phases with back tracing (iteration) between the phases. These five phases and corresponding activities are:

- Analysis – determine the scope of the problem, determine the requirements for the new system, generate the software/system requirements specification (SRS) document.
- Design – define the precise functionality of the software/system, describe constraining algorithms, determine interfaces between software modules, produce detailed documentation of complete software/system design.
- Implementation – using the design document, transform the described functionality and algorithms into "computer-understandable language" [13].
- Testing – initially test each module (unit test) as it is implemented. Subsequently integrate the module into one working unit and test each module and the overall system as the modules are introduced to the system. Finally, test the complete system after it has been installed in the working environment.

- Maintenance – continuously fix errors as they are found, add new functionality based on additional customer requirements, and enhance system performance as necessary through improved modules.

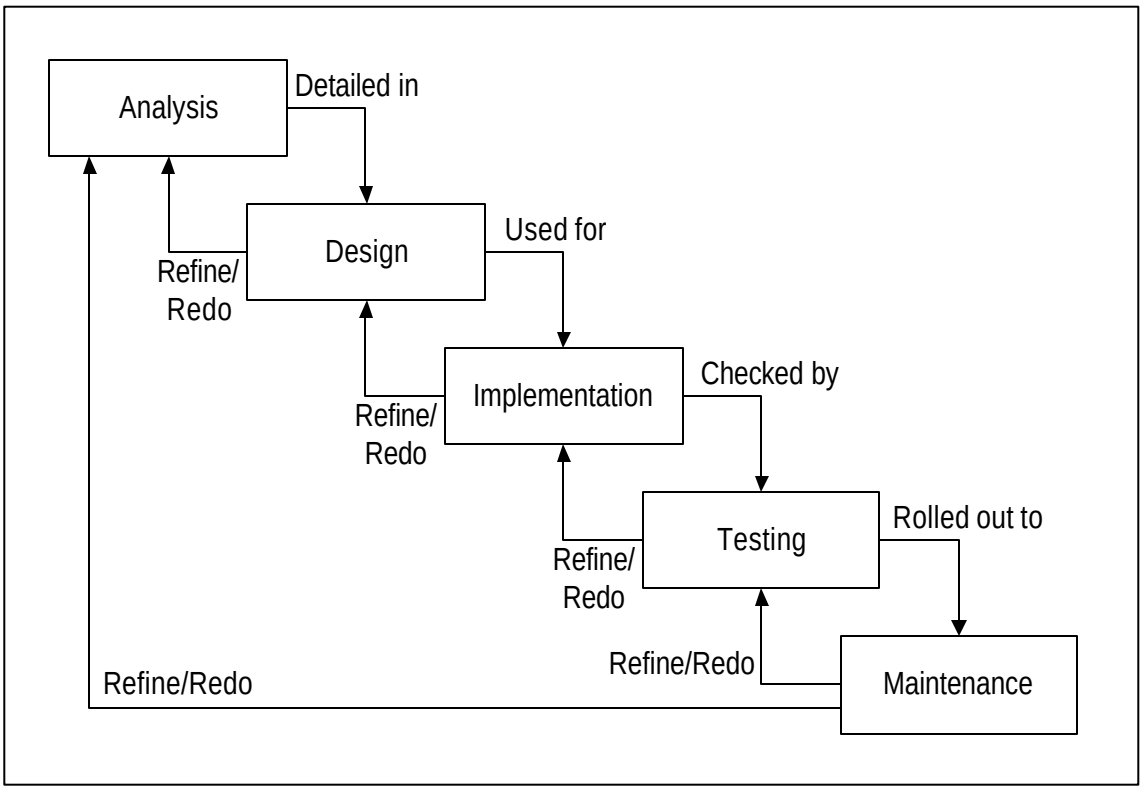



**Figure 2-1.** Waterfall Life Cycle Model

Iterations between each phase allow for enhancing the product, or fixing problems discovered during phase transitions. Interestingly enough, for many years after Royce introduced the Waterfall model, most software development companies thought of it and used it as strictly linear [34]. Recently, possibly due to the successes of other iterative (incremental) models, the Waterfall model has been acknowledged with iterative "loops," now often called the "Extended Waterfall Model." The major contributions of the Waterfall model can be found in 1) a phased approach to the SDLC, 2) major, distinct activities identified for each phase, and 3) using iteration to further develop or refine the project artifacts. The benefits to requirements from these contributions can be realized in that 1) an acknowledged phase exists specifically for the generation of requirements, and 2) through iteration between phases, requirements, as well as all other artifacts, are viewed as needing refinement throughout the process.

*Spiral*

The original Spiral model (Figure 2-2) as defined by Boehm [6] consists of six task regions: customer communication, planning, risk analysis, engineering, construction and release, and customer evaluation. Each project phase, from concept definition to maintenance and future enhancements, iterates through the six task regions. Specifics of the regions are:

- Customer communication – establish the scope and requirements together with the customer.
- Planning – analyze the requirements from the customer and determine possible solution approaches
- Risk analysis – together with the customer, determine the risk involved in each alternate solution approach. Customer may decide on approach with appropriate risk level.
- Engineering – create the design of the system, and describe the necessary algorithms.
- Construction and release – code the software based on the engineering design. Also conduct unit, integration, and system testing and release product.
- Customer evaluation – determine if product is satisfactory and what changes or enhancements are necessary.

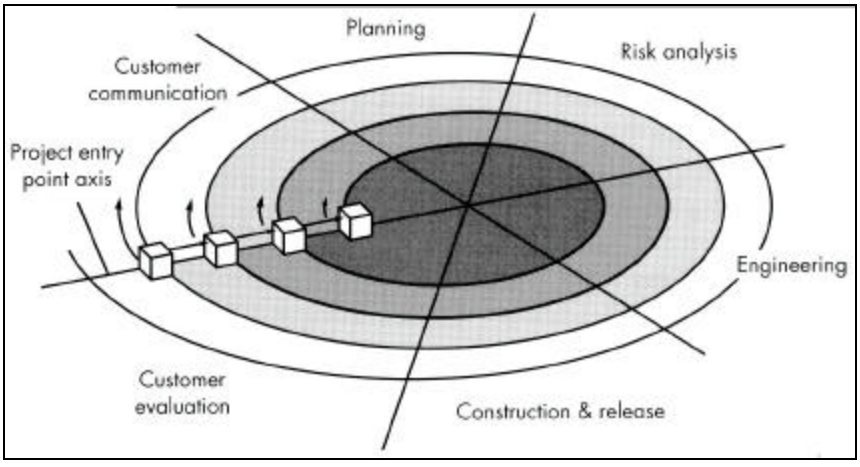



**Figure 2-2.** Spiral Model [34]

Although we have only described a single iteration through the six regions, the Spiral model most often is deployed as an incremental model. This means that the software/system is constructed through multiple iterations until a final product is reached.

A strong focus of the Spiral model is on communications with the customer. Three of the six phases require the software engineer to work together with the customer to 1) decide on a system, 2) determine the risk of the system, and 3) evaluate the delivered system. Boehm's emphasis on communication, the importance of involving the customer, plays an even larger role in later variations on the Spiral model, such as the WINWIN Spiral model which includes regions for negotiations with the customer [7]. Furthermore, refining the system/software through an iterative approach allows for the customer and software engineer to remain focused on smaller, manageable issues within each iteration. We would expect this to enhance the quality of the final product. Not only because this high degree of customer to software engineer communications is expected to provide a much more refined and customer expectation meeting set of requirements.

*Prototyping (throwaway and evolutionary)*

Prototyping evolved out of the necessity for rapid illustration of the human interfaces and interactions while the system is under construction. Rapidly developed prototypes would not be used in the final system and, depending on the prototype system used for development might have little or no underlying functionality. This prototype was called a throwaway, since little cost and time is spent to design and refine the system interface, and the prototype is discarded. The information gained from the throwaway prototype is used to build the actual system with a "feel and touch" reflecting the user's needs.

The evolutionary prototype differs from throwaway in that the initial rapidly developed "show-and-tell" system is not discarded. Rather, through multiple iterations the prototype is expanded and refined to yield the system meeting the user's needs. A challenge faced by developers using the evolutionary prototype approach is keeping the code base efficient and free of errors, while continuously adding new code or making changes to

existing code. Good programming style is necessary to avoid producing a system that is inefficient and error prone, possibly resembling “spaghetti code.”

The major benefits of prototyping for software development are found in rapidly enabling the customer to get a “touch-and-feel” for the human-computer interface that the system provides. Furthermore, it allows the software engineer to provide the customer with an early selection of interface alternatives, which can easily be changed to the needs of the customer, as well as to help refine “fuzzy” requirements or requirements that are not well understood. Initial prototypes can be created with little expense, possibly re-using existing interface designs. The benefit the software engineer gains with this approach is the understanding of how the customer “likes” to interact with the system. This is valuable information not only for the actual software implementation, but also for aspects of the software’s functionality that might be influenced by the interface. Using prototyping therefore should enable the software engineer to create a requirement specification with precise details about the user interface and expected functionality of the system.

*Extreme Programming*

Extreme Programming’s (XP) objective is to shorten the development time of a project, while increasing the interaction with and feedback from a project customer [5]. XP advocates sidestepping many of the “traditional” software engineering activities, i.e., generating requirements, analyzing, and designing, and focusing primarily on implementation.

XP embraces daily incremental test case development, daily coding, daily compilation - all of these based on new, daily-learned requirements from the customer. Without doubt, Beck opened himself up for much criticism from the “traditional” software engineering community by eschewing the early stages of development universally accepted as necessary. However, rather than judging XP as good or bad, model, method, methodology, or a hacker’s paradise, we want to focus on those aspects of XP that parallel those of other models, or that we believe can enhance the activities of generating requirements.

As with other models and approaches, communications and customer involvement are a major factor of XP. But unlike other approaches, XP takes this to the "extreme" by communicating with the customer at every incremental successful code compilation. Furthermore, XP developers have to gather new requirements incrementally from the customer after each code increment. These increments usually are no further apart than a week, often only two or three days. Should the customer have a change in requirements, these can be included immediately from one increment to the next.

Evaluation is a second important aspect of XP. Evaluation is done by 1) writing test cases before coding, 2) testing code based on the test cases after each incremental code compilation, and 3) submitting incremental code releases to the customer for review. Also, XP includes a novel approach to coding called pair programming. In pair programming, two developers always work together, one developer on the keyboard and the second giving instructions or simply reading over the shoulder of the typist to ensure coding accuracy. In a sense, this is a fourth type of evaluation, one that is continuous during implementation.

Although XP is not generally accepted as a "tried and true" methodology, we believe that the concentration on customer involvement and various continuous evaluation methods being applied have a positive effect on the end product. These pieces, if taken into a different context and adapted to a more "traditional" process would, without doubt, be beneficial to software development. The impact XP has on requirements generation, and moreover, documentation of requirements remains to be seen. Since it is the developers that receive requirements directly from the customer, the requirements documentation produced might be much closer to code or design documentation, specifying how, rather than what is implemented.

*Cleanroom Software Engineering*

In an attempt to create a formal specification and verification model, Mills introduced the Cleanroom Software Engineering (CSE) model [31]. CSE consists of nine steps, which are iterated (therefore an incremental approach) [34]:

1. Increment planning – decide on the set of requirements, the necessary project size, and a delivery schedule for the current increment of the project.
2. Requirements gathering – create a very detailed description of the customer's requirements.
3. Box structure specification – using black box, state box, and clear box methods, describe each requirement in a functional, formal (mathematical) specification.
4. Formal design – by extending the box structure specification, connecting outputs with inputs on the boxes, a formal design is created.
5. Correctness verification – the formal design is evaluated using a top-down approach, from high-level to low-level boxes. This is done through "correctness questions" [28] that are applied to the design.
6. Code generation, inspection, and verification – the formal design which exists in a "special" CSE language is now translated into code. This code is subjected to code walk-throughs and other code verification techniques.
7. Statistical test planning – test cases are designed that take into account the projected usage pattern of the software product.
8. Statistical use testing – using statistical quality control calculations, the software is tested for different use scenarios.
9. Certification – after all tests are successfully completed, and all errors found are fixed, the current software increment is "integration ready" certified.

Each step of the Cleanroom approach is formal. This formal approach allows for statistical control of the quality, not only of the approach taken, but also of the artifacts produced within each step. The last step, Certification, can be achieved because the product developed has been mathematically verified and its reliability can be quantitatively stated. Software that is developed using the Cleanroom Software Engineering approach has a low rate of failure that cannot be achieved by more informal approaches [34]. The drawback to this approach is that to benefit fully from the formal methods, most project members have to be highly skilled in the Cleanroom approach. This skill level among an entire team takes years of experience and rigor to achieve, and requires little or no attrition to be maintained. This alone makes it very hard for any software development com-

pany to deploy the Cleanroom approach. Furthermore, the formal approach adds additional time to the development schedule; one could argue that the additional development time, exponentially increasing with project complexity, results in a final cost too high for customers to pay. Again, the Cleanroom approach is faced with the same dilemma as other approaches that add more rigor and quality to the process, the high level of quality they can produce has to take a backseat to the amount of money and time a customer is willing to spend on the project.

### 2.1.2 Methods

This section focuses on software engineering methods designed to provide a specific solution or achieve a pre-determined goal in the software development process. Methods, unlike models, are often more focused on a single problem within the software development process, rather than providing an overall process approach. This does not mean that they are limited to a single phase of the development process, but rather, that they can be applied multiple times throughout the development process. The method is applied when the situation (or problem) it is designed to solve is encountered. Some of the methods we focus on are continuously applied throughout the SDLC to prevent problems from occurring, instead of solving problems when they have occurred. The methods we focus on are the Fagan Design Inspection Process [18], Divide and Conquer [33], Verification and Validation [27], Structured Analysis and Design Technique (SADT) [37], and finally, Facilitated Application Specification Techniques (FAST) [34].

*Fagan Design Inspection Process*

Michael Fagan was a software engineer working for IBM in the 1970s. He was tasked with developing a method to ensure quality software design documentation that would decrease the cost IBM was spending on fixing problems in code, or maintenance traceable back to erroneous design. The method he created is known is the Fagan Design and Code Inspection Process [18]. Fagan evolved the inspection process to not only be used for design and code inspections, but for all artifact inspections during the SDLC. Fagan now has his own company Fagan Associates [19], teaching software development companies how to apply the inspection process in the SDLC.

The Fagan Inspection Process structures an otherwise all too often ad hoc meeting to review project artifacts. This structure consists of two areas, the process and those involved in the process. The process consists of:

- Several well-planned meetings and sessions, including planning, overview, individual preparation, inspection meeting, re-work, and follow-ups.
- Materials guiding one through the process, such as individual preparation logs, defect summary logs, defect trouble report, and notebooks for each inspector to track problems.
- Guidelines for the amount of time that should be spent during each meeting or session, how follow-ups and re-work should occur, and who should be involved at any stage of the process.

Those involved in the inspection process, the inspection team, are assigned specific roles that they must follow. These roles are defined as:

- Moderator: responsible for gathering and distributing all inspection artifacts, and leading the inspection team throughout the process and meetings
- Reader: responsible for carefully disseminating the artifacts and presenting them in a logical consistent manner during the inspection meeting
- Recorder: records all defects found and, with the help of the inspectors, classifies the defects by category and severity; helps the moderator in assembling inspection reports.
- Inspector: carefully investigates the assigned artifacts during the individual preparation period of the inspection process, and presents findings during the inspection meeting. Every participant in the inspection meeting plays the role of inspector, including the artifact author if this person is included in the inspection process.

The focus of the inspectors during individual preparation, as well as during the inspection meeting, is to uncover artifact defects that could be classified as: ambiguous, inconsistent, incomplete, not testable, and not traceable. Furthermore, inspectors should look for overall artifact structure and imposed standards the artifact should be based upon. Any

defect, critical or non-critical, must be logged and presented during the meeting. The overarching responsibility of inspectors is the detection of all defects, without providing solution suggestions to those defects.

The benefit of the Fagan Inspection Process is that multiple people inspect the same artifacts. This allows for uncovering a high percentage of errors. Furthermore, the necessary re-work and follow-up ensure that errors do not propagate to later SDLC phases. This has a tremendous impact on the overall cost of fixing errors. As Barry Boehm shows, each undetected error moving from one phase to the next increases the cost of fixing ten-fold [8]. Therefore, the Fagan Inspection Process contributes to project cost savings.

The disadvantage of the inspection process, as is true for any process or method that adds more rigor, is the fact that more time must be expended to complete the inspection process. Depending on the rigor that the development team must use, there may be no delay due to the inspection process in proceeding from one phase to the next, or the inspection process may be used for stop-go decisions to continue between phases. In the former case, the inspection process findings lead to corrections in artifacts from the previous phase or phases, as well as possible corrections of artifacts in the current phase based on previous errors. In the latter case, all errors detected in the latest inspection process must first be corrected in all artifacts from previous phases before new work can begin.

As is the case with any process, the paying customer must decide on how much time and money they want to expend on conscientious development. That determination must include maintenance to fix those errors not detected early (which we assume the customer wants to avoid, while hoping that any savings achieved through shortcuts up front will not lead to noticeable problems later, but to an overall cost savings for the product).

*Divide & Conquer*

Divide and conquer is as much a method as it is a concept. As such, it does not stem from one particular area of software development to solve a problem - it most likely does

not have its original roots in software development. The method used by divide and conquer though parallels that of successive refinement. The concept itself is a simple one. Given a large problem, divide this problem into smaller, more easily solvable problems, and solve each small problem on its own. When each problem has been solved, add the solutions together to yield the overall solution to the initial large problem. Smaller problems that are still too large to solve may be further sub-divided. The difficulty is to determine how many times a problem should be divided, and where to draw the dividing lines. References in software development to Divide and Conquer are usually found in data structure books in the sections on sorting [44], in graphic design books in sections on how to draw lines or shapes on the screen [20], and in formal language theory books [12]. There are no direct references to Divide and Conquer in software engineering literature. Yet, we are introducing the concept of Divide and Conquer as part of the general SDLC because many of the approaches we take within the software development processes are based on Divide and Conquer. For example, in defining requirements for a project, we start with an overview of the entire project and, next, proceed to define and refine pieces of the project, until we have all requirements defined. The same is true for design, which is often split into a high-level and a low-level phase; but even within each phase we initially look at the overall problem, and then proceed to design the individual pieces. The Divide and Conquer concept is ingrained in both top-down as well as bottom-up coding approaches.

As such, Divide and Conquer allows software engineers and others involved in the software development process to better understand and solve smaller problems, which ultimately, through iterative joining, lead to the large problem solution. The difficulty introduced here is that when working on a small problem, one must always be aware of the larger problem that should eventually be solved. Furthermore, the interaction between solutions (such as modules) that must interface as part of the larger solution must be well understood and examined to function correctly together when integrated.

*Verification and Validation (V&V)*

Verification and Validation (V&V) is a method common throughout the entire SDLC enabling the software engineer to produce quality software development artifacts, including the delivered product [27]. Verification of development artifacts ensures that the right product is being developed. Verification of artifacts is conducted between each SDLC phase, as part of the transition between phases. Validation of the product ensures that the right product has been developed. This means that validation takes place after the product has completed integration testing and is validated back to the original concept or software requirements specification document. Robert Lewis in his book [27] provides checklists and questions to ask during the verification and validation process.

From the abstract, the concept of V&V and the Fagan Inspection Process seem very similar. Indeed, from the standpoint of ensuring software quality, they are. Unlike V&V though, the Fagan Inspection Process concentrates on the structure of the process of "verifying" artifacts. Fagan describes the meetings that have to take place, the participants in the meetings, and provides logs to track defects. V&V, on the other hand, does not prescribe meetings that need to take place, or who should be involved. Instead, V&V focuses on the artifacts that need to be verified and describes how these should be verified. Guiding this are the checklists and questionnaires provided by the V&V method. Therefore, the V&V activity can include parts of the Fagan Inspection Process to help structure the actual inspection meetings. This combination of V&V is something we find in Software Inspection Process [16], as well as in Software Inspection [21]. The National Aeronautic and Space Agency (NASA) has adopted a V&V methodology called the Software Engineering Evaluation System (SEES) [15], which is also a combination of V&V and the Fagan Inspection Process. In addition to the elements of V&V discussed above, SEES also includes a large amount of additional documentation guiding the software engineer through the SEES process, as well as tracing requirements from concept through code, with the ability to completely classify and track defects for each requirement discovered.

Lewis describes the IV&V methodology in his book [27]. The "I" stands for independent, meaning that V&V is not performed by those who created the original artifacts, or even by another "QA" group within the same company. Rather, V&V is made independent by moving tasks outside of the realm of those producing the software product. This is beneficial to customers who require independent certification that the product they are procuring performs as stated by those producing the product. Internal QA often has to give in to the pressure of marketing or delivery schedules promised by management, and therefore QA concerns are disregarded. This can lead to products with false certification of its abilities. An independent group reports only to the customer, and as such has no interest in the pressures of the development company. The best IV&V takes place when the IV&V team does not know the development company or vice versa, and all communications of defect findings are done through the customer. As with any extra-added effort to ensure software quality, the customer must decide if the increase in quality is warranted by the extra time and money spent.

*Facilitated Application Specification Techniques (FAST)*

FAST is an approach to bring software engineers and customers together in meetings, focusing on identifying the problem, suggesting solution approaches and alternatives, and developing a set of initial requirements [34]. Key factors of FAST are:

- Meetings are held at neutral sites, not the software engineer's or customer's facility
- Rules of engagement are used during the meeting
- Facilitator conducts the meeting
- Goals to be accomplished are: identify problem, suggest and negotiate solution approaches, and define preliminary requirements

Some previously described approaches work well within the confines of FAST. Joint Application Design (JAD) [11], as described in Chapter 1, is one such approach. A well-facilitated JAD session can quickly yield much information on the project, the requirements, and the risks involved. One reason these sessions help uncover large amounts of information is that participants' contributions are visible to all, which allows other par-

ticipants to recognize relationships to yet more problems or requirements, thereby creating an environment that instigates the generation of ideas.

The drawback to JAD sessions can be found in the manner in which participants interact, and the reporting structure between participants. Negative comments about a participant's contribution can effectively "turn off" any further contributions from the "attacked" participant. Managers "pulling rank" may lead subordinates to think their contributions worthless, and again hinder further active participation. Therefore, successful JAD sessions are set up with the understanding that all contributions must be considered valuable and that ranks and levels, or evaluations have no place in the JAD session.

A second approach to customers and software engineers working closely together during the SDLC is called Participatory Design (PD) [10, 17, 22, 29]. Unlike JAD, which relies on the customer and software engineer to work together during several FAST meetings, in PD, the customer and software engineer stay closely involved during the entire SDLC. This close working relationship prescribed by PD has the benefit that the knowledge gaps between the customer and software engineer are lessened or closed. The customer will learn details of the software engineer's job. Similarly, the software engineer can learn about the customer's current environment and job function and better understand the needs of the new product being developed for the customer as enhancement or replacement for the current system. This close working relationship results in a product that 1) meets the customer's needs, 2) includes the customer's design ideas and therefore is embraced by them, and 3) relies on the customer and software engineer for qualitative testing throughout the development process. Mambrey writes that a major drawback to PD can be found in the politics of a customer's company [29]. Management is often not involved in the relationship with the software engineer. Rather, the end-users work closely with the software engineer. This leads management to believe that their needs are not considered as part of the development process and final product. Furthermore, management is often not willing to let end-users spend the necessary amount of time required with the software engineer, as this affects overall productivity. As with the previously described "quality enhancement" approaches, PD is commonly accepted as a develop-

ment method, but is contingent upon the extra time and money that can be spent during development.

The common goal of each of these methods is the increase in quality of the software product. Each method takes a different approach, but all involve a thorough examination of the work product, either by direct examination, or by involving other participants to gather more knowledge about the product. And, each method has the benefit of increased quality if applied. Yet, those paying for the product may see the extra personnel and additional time involved as a disadvantage.

The methods we have described are all considered to have a positive impact on the generation of requirements. Fagan Inspections as well as the V&V techniques directly examine the existing set of requirements; defects found must be corrected before development continues. FAST as well as Divide and Conquer methods are used even before the examination of requirements are undertaken, contributing to the direct definition process of requirements. With FAST the expectation is to gather requirements quickly from a set of stakeholders, while at the same time exposing possible contradictory needs. The Divide and Conquer method enables splitting larger, more difficult problem domains into smaller, easier to handle ones, thereby, helping structure requirements around small, easy to understand and explain problems. At the same time the common interfaces between the split problem domains are highlighted.

## 2.2 Requirements Analysis

When the task of gathering requirements is complete, these requirements must be analyzed. This phase is therefore referred to as the requirements analysis phase. The objectives of requirements analysis are to:

- examine the set of requirements for several quality attributes, including consistency, unambiguity, testability, and the overall completeness of the requirements
- create “functional” requirement groups
- develop a comprehensive requirements specification document

Achieving these objectives necessarily requires communication with the customer, to improve on the quality attributes of the defined requirements, as well as to follow up to define "missing" information. This is an iterative process that results in the software requirements specification (SRS). Upon completion of the SRS, the requirements analysis phase is complete and the SRS is an artifact moving forward into the high-level design phase.

Unlike the requirements analysis phase we describe here, and as is similarly described in textbooks [33, 34], the term "requirements analysis" is often misused to describe the entire requirements phase. No distinction is made between problem identification, definition of requirements, and analysis of requirements. Furthermore, the major focus remains on those activities that are required for meeting the objectives we describe for requirements analysis alone. This strong emphasis on the analysis of requirements, and the subsequent inspection of the requirements document to ensure the quality attributes of the gathered requirements, unfortunately highlights how little significance is given to the initial generation of requirements. The following section identifies tasks and artifacts of the general requirements analysis process as used by most software development efforts.

### 2.2.1 Requirements Analysis Process

From an abstract point of view, the standard requirements analysis process begins when a customer expresses the need for software or system development to the software engineer. This need is usually expressed in a business concept or scope document. The software engineer's task then becomes taking the conceptual idea and identifying necessary requirements to realize those ideas. Upon completion of identification and definition of requirements a transformation of often-unstructured lists of requirements into a software requirements specification (SRS) document is the next step.

The approaches taken to transform requirements into the SRS document range from informal to very formal. The informal approach takes the list of requirements, defines a relationship between the different requirements to determine a proper grouping, and produces a document written in natural language. The most formal approaches take the

same list of requirements and create an equivalent mathematical formal notation of the requirements to enable establishing correctness and completeness of the requirements. No matter which approach is taken, the amount of time spent on analysis of the requirements in creating the SRS document is often much longer than the time spent identifying the requirements. We have introduced the most common methods used in analysis in this chapter.

Once the SRS document has been completed, it is assumed to clearly identify the needs of the customer, expressed in semi-technical terminology. The assumption is that the wording of the SRS is unambiguous, consistent, and that all functional requirements are testable. The National Aeronautics and Space Agency's (NASA) Software Assurance Technology Center (SATC) at the Goddard Space Flight Center in Maryland has developed the Automated Requirements Measurement (ARM) tool to parse an SRS for key quality indicators of the written requirements [43]. Besides counting lines of text and words, the tool searches for key words indicating requirements (e.g., "shall," "must"), as well as indicators that show weakness in the requirements (e.g., "not limited to," "see attached") and optional requirements. Using the information provided by the tool, a software engineer could actively improve the SRS. As Rosenberg states, the ARM tool does not provide information regarding the actual correctness of the stated requirements [43]. One could assume though, that a side effect of the ARM tool is an improvement of requirements documentation over time and in future projects. Those using the ARM tool continuously learn what mistakes are made and can avoid them in future documentation.

### 2.2.2 A Critique of the Existing Requirements Analysis "Processes"

The models, approaches, and methods presented in this chapter have a common goal: improvement of software requirements. The direction they take extends from the very informal to the highly mathematical formularization. Furthermore, some work towards single focus objectives; whereas, others try to provide an environment to improve the overall quality of the software produced. However, with so much emphasis on requirements improvement, we need to raise the question: why do 72% of projects still continue

to fail because of requirements-related issues [24]. The available approaches and methods to generate high quality requirements are plentiful.

What is lacking in all these approaches and methods is a common guiding structure, a structure that provides a visible breakdown of the individual tasks and timelines in generating requirements. Furthermore, most approaches are stand-alone approaches, with very limited focus and benefit. For high-quality requirements to be generated, these approaches need to be bonded together under a framework, with common interfaces between the artifacts. But, a binding framework and structure are not the only missing elements in improving requirements. An understanding of the importance of each aspect of requirements generation is needed. Many approaches and methods deal with improving and analyzing requirements. Experience over the past several years supports the estimate that 80-90% of all effort on requirements is spent on analyzing and improving what has somehow been received from the customer. That leaves 10% for gathering requirements from the customer. Therefore, very little time is spent actually working with the customer to determine and adequately define not only their needs but also intentions. That the end product, created from “flawless” requirements, fails to meet the intentions and needs of the customer should not be surprising.

In the next chapter we introduce a framework that refines the phase we all too often mislabel “requirements analysis.” Our focus is on the requirements definition phase that precedes requirements analysis, focusing on gathering the correct requirements from the customer, thereby meeting customer intent. We introduce elements that help structure and necessarily constrain the software engineer and customer within this phase. Furthermore, in Chapter 4 we introduce a monitoring methodology providing procedures and methods to the software engineer that confine not only activities within the constraints of the framework; enable detection of problems in the requirements elicitation process, and offer solutions to these problems. Our framework and methodology are comprehensive to define high quality requirements together with the customer, yet flexible enough to allow for extension and inclusion of other approaches or methods depending on the development environment.

# Chapter 3. Requirements Generation Model

## 3 REQUIREMENTS GENERATION MODEL (RGM)

The objective of this chapter is to introduce the Requirements Generation Model (RGM), and components of the RGM. In the closing of Chapter 2 we discuss the need for a comprehensive structure to the requirements generation process. A structure that not only clearly defines the requirements phase, but also includes sub-phases with specific goals and objectives. Furthermore, we believe that through a well-defined structure, methods and approaches to improve requirements can be tied together.

The scope of our work within requirements generation is focused on defining requirements, the first component of requirements generation. Truly, Requirements Definition is a separate phase with an iterative component, preceding Requirements Analysis. The artifacts produced during Requirements Definition enter into "Requirements Analysis" when pre-defined exit criteria are met within the Requirements Generation Model (RGM). The overemphasis on requirements *analysis* within the software and requirements engineering community leads us to focus on the often neglected area of requirements definition, trying to understand and record customer intent. We perceive the distinct splitting of the "requirements phase" into two separate phases with clearly identifiable objectives for each, similar to the splitting of the design phase into high-level and low-level phases. Here, splitting the design phase into two separate sub-phases helps to focus on the distinct tasks, activities and artifacts that are expected of each. Furthermore, artifacts from the high-level design phase are transitioned into level-low design for further refinement to create the "build-to" design specification document. Similar to this flow in the design phase, we expect the artifacts produced through requirements definition in the RGM to initiate and drive activities during requirements analysis.

The RGM is based on two components, 1) a framework that structures and controls the activities within which the customer and the requirements engineer should define requirements, and 2) a monitoring methodology that ensures that all requirements elicitation activities follow proper procedures. The focus of this chapter is on the Requirements

Generation Framework (RGF) of the RGM. The monitoring methodology is discussed in the following Chapter 4.

### 3.1 Requirements Generation Framework (RGF)

The RGF consists of a structuring component and a constraining as well as a guiding component. The structuring component takes the conventional requirements analysis phase and partitions it into an indoctrination phase and an iterative requirements capturing phase as seen in Figure 3.1. The requirements capturing phase is further sub-divided into preparation, elicitation, and evaluation sub-phases. Furthermore, a requirements management contingent spans all phases of the RGM. Requirements management is an important aspect of the RGM as it enables the requirements engineer and customer to succinctly track defined requirements, control revisions to requirements, and in moving towards design allow tracing of each requirements to their design components.

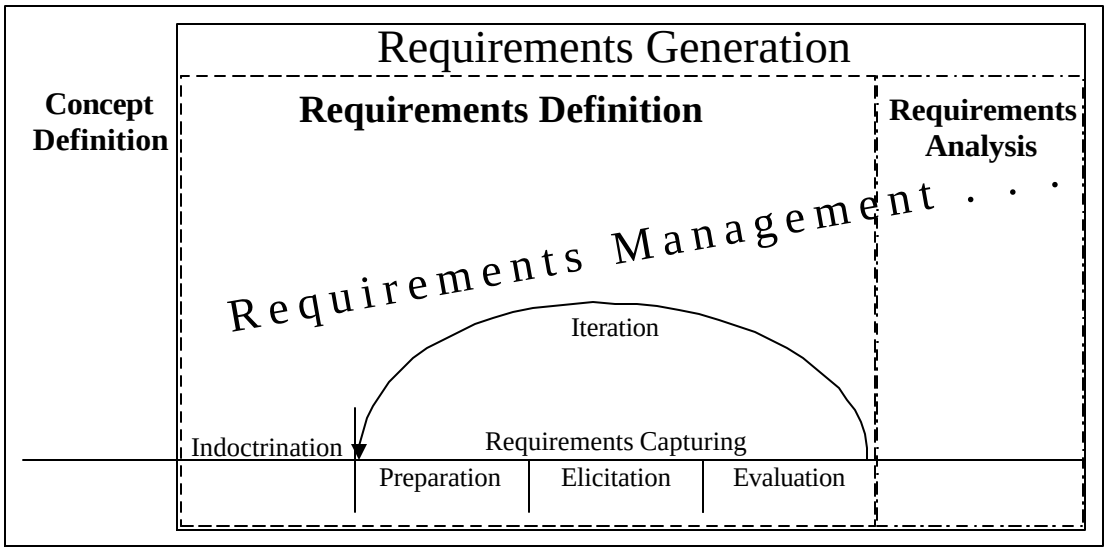



**Figure 3-1.** Requirements Generation Framework

The constraining and guiding components of the RGF are provided by a set of protocols and guidelines respectively. Protocols establish boundaries for the RGM within which the customer and the requirements engineer must operate. An analogy is that protocols are like guardrails at the sides of a road. A car must stay within these guardrails to safely continue on the road. Equally, the customer and requirements engineer must stay within the constraints of the protocols to be considered operating within the RGM. A set of

protocols developed for the RGM is presented in Appendix A. We reference some of these protocols in later discussions in this chapter. Guidelines, unlike protocols are not constraining and need not be adhered to by either customer or requirements engineer to operate within the limits of the RGM. Rather, they offer suggestions on how to further refine some of the operational aspects of the RGM. Following guidelines is expected to enhance the overall ease of the workflow of the RGM. A set of guidelines is presented in Appendix B, and examples are provided as discussion points in this chapter.

The next sections discuss in detail each of the phases and sub-phases of the RGF together with the objectives and activities designed to achieve these objectives.

## 3.2 Indoctrination

The objectives of the indoctrination phase are:

a) familiarizing the customer with the RGM,
b) introducing the requirements engineer to the customer's domain, and
c) defining and setting up tasks and responsibilities required during the requirements capturing phase.

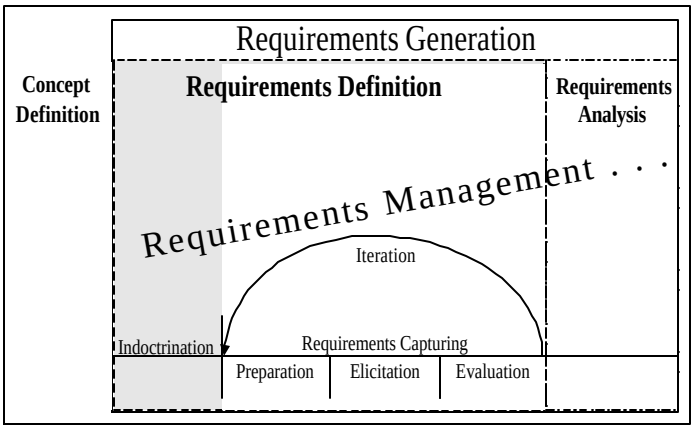



Figure 3-2. **RGF Indoctrination**

This phase is crucial in preparing both the customer and the requirements engineer for the joint efforts required in defining requirements based on the customer's intent, as well as their continuing relationship through the rest of the development lifecycle. We examine each of the objectives of the indoctrination phase next.

### 3.2.1 Educating the customer (guideline)

An important aspect to generating a good set of requirements is the familiarization of all participants in the RGM process. Familiarity with the RGM allows the participants to understand the workflow through the RGM, know when protocols are not followed, and

successfully apply the monitoring methodology. To that extent we recommend that the requirements engineer educate the customer in:

1. The software development lifecycle – shortly describing the phases through which the development of the customer's product will progress, with special focus on the role of the customer in each phase.
2. The requirements engineering process – here the requirements engineer should focus on requirements, particularly describing what makes up a requirement, and what does not represent a requirement.
3. The working relationship – paying particular attention to the responsibilities the requirements engineer and customer must assume.
4. The requirements generation model and framework – in preparation for the immediate upcoming activities.

We understand the constraints of time and cost on projects, as well as possibility of a customer not wanting to be directly or as much involved in the process. Therefore, the educational part of the customer in the indoctrination phase is not a mandatory activity, put considered a guideline of the RGM.

### 3.2.2 Educating the requirements engineer (protocol)

Educating the requirements engineer is a responsibility the customer must fulfill. Unlike the previous activity, this activity is mandatory, considered a protocol of the RGM. The objective of educating the requirements engineer is to provide them with the best possible information about the current system and how it provides for the customer's needs, as well as an understanding of the needs and functionality of the proposed system. To achieve this objective the customer should provide the requirements engineer with:

1. A presentation of the current system (if one is in existence), including the environment in which it is installed, and an explanation of what needs are currently fulfilled by the system.
2. An introduction to the needs that the proposed system should accommodate. During this introduction the requirements engineer should learn the environment in which the system will be installed, including any interfaces or special concerns for pre-existing system with which the proposed system must interact.

3. A discussion focusing on the motivation for changes to the current system or way the task is performed, outlining the purpose and overall scope for the new system. Furthermore, during this discussion the requirements engineer should learn who are the possible stakeholders in the new system.

The overall goal of educating the requirements engineer is to give them as much preliminary information about current and proposed system to allow them to ask relevant questions of the correct stakeholders as the requirements are defined.

### 3.2.3 Tasks/Responsibilities/Preparation

The final activity of the indoctrination phase focuses on educating all participants in the process about their tasks and responsibilities. Throughout the requirements elicitation meetings, participants may be assigned varying responsibilities with different tasks to be performed. The participants must be aware of the responsibilities that come with each role, especially the preparatory tasks that lead to the requirements elicitation meetings. In particular, responsibilities for certain activities are assigned to either requirements engineer or customer based on their domain knowledge, such as:

- The requirements engineer is responsible for ensuring adherence to the RGM process. This is the case since the requirements engineer is most familiar with the requirements generation process.
- The customer is ultimately responsible for ensuring that the final set of generated requirements truly reflects the needs and intent of the proposed system. The greatest knowledge of the customer's domain and the required functionality of the new system lies with the customer, therefore this assignment of responsibility.

The requirements engineer is expected to stress the importance of ownership and responsibility for completing all assigned tasks prior to any elicitation meetings, and to maintain open communication between all participants. Preparation and readiness is of utmost importance since requirements are mainly identified during the elicitation meetings.

In undertaking the preparation for the first requirements elicitation meeting, several items most be considered:

1. A requirements management tool must be identified and set up to facilitate the recording, analysis and change control of all defined requirements after each elicitation meeting.
2. The preliminary objectives and agenda for the fist meeting must be identified and published.
3. Participants and issues meeting the objectives must be identified and published.

The activities identified in this last objective of the indoctrination phase are considered mandatory, and therefore are a protocol of the RGM. When these indoctrination activities have been completed, the project can proceed forward into the preparation sub-phase of requirements capturing within the RGF. The preliminary activities discussed in the section prepare the requirements engineer and customer for forthcoming meetings, and should aid in mitigating some of the domain and technology knowledge gap that exists between them. Through these educational sessions the customer learns about the development and elicitation process and the requirements engineer gets to better understand the customer's domain and needs for a new system. This learning contributes to a more effective working relationship as the project proceeds through the requirements capturing phase.

### 3.3 Requirements Capturing

The requirements capturing phase of the RGF follows the indoctrination phase (Figure 3-3). This phase consists of the three sub-phases: preparation, requirements elicitation, and review. The focus of each of these sub-phases is on 1) preparing hfor the upcoming requirements elicitation meeting, 2) conducting the elicitation meeting, and 3) reviewing and analyzing the artifacts defined during the immediately preceding meeting, respectively. As illustrated in Figure 3-3, these three sub-phases are iterative, allowing for re-

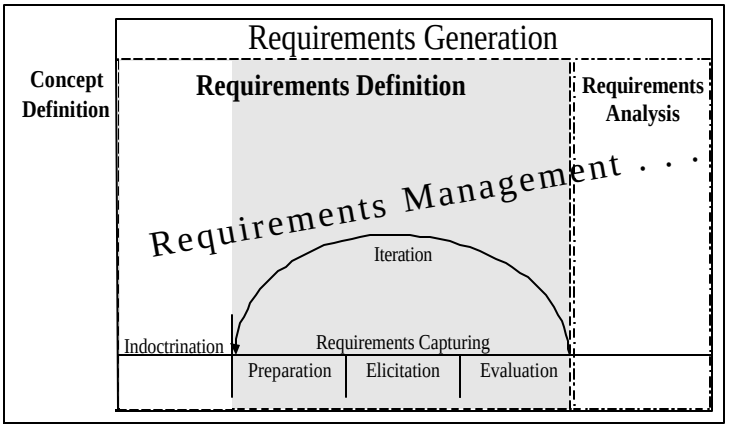



Figure 3-3. **RGF Requirements Capturing**

quirements to be progressively elaborated and refined. A checklist of exit criteria (Appendix D) allows the requirements engineer to determine when no further iterations through the requirements capturing phase are required. The following sections discuss the details of each sub-phase, presenting the objectives and related activities that ultimately provide for the structure of the framework and the process to effectively define customer requirements.

### 3.3.1 Preparation sub-phase

The objectives of the preparation sub-phase are a) for participants in the upcoming elicitation meeting to review and resolve issues, b) to schedule and organize the elicitation meeting and c) to maintain a high level of communication between all participants. Besides the requirements engineer, the participants from the customer arena can consists of managers, analysts, users, and others as necessary. Most participants in the elicitation meeting will have been assigned tasks that should be completed during this phase. These tasks may have originated from either a) initial issues identified during the indoctrination phase, or b) the review conducted after prior iteration through the requirements capturing phase. We now examine each of the three objectives of this phase.

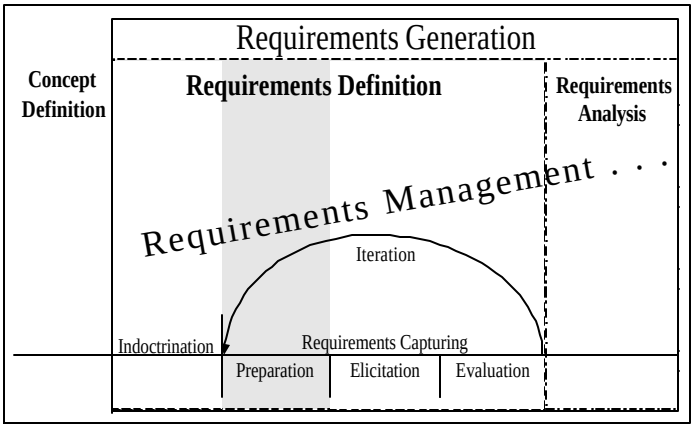



Figure 3-4. **RGF Preparation**

*Review and Resolve Issues*

As stated previously, issues that must be resolved can stem either from the indoctrination phase or from a previous iteration through the requirements capturing phase. This is due to the preparation phase having two possibilities for entry, as shown in Figure 3-1. Regardless of where the issue is initiated, during the preparation phase we ideally expect the person assigned this issue to find a resolution. However, it is highly unlikely that all issues can be resolved in the time given for preparation, without slowing down, or holding up the project from progressing. It is possible that an issue requires further exploration of

other subjects related to it before it can be resolved. These subjects may not yet have been explored during prior elicitation meetings. Therefore, the issue stays open until all necessary information is available for it to be resolved. Other issues may require information from stakeholders or subject matter experts that are not readily available. Again, the issue may be put on hold until resolution is possible.

It is the responsibility of the requirements engineer to track all open issues and to determine when these have been adequately and sufficiently resolved. The requirements engineer may need to consult with the customer regarding adequate resolution of some issues that are outside of their domain knowledge.

*Scheduling and Organizing the Elicitation Meeting*

The requirements engineer continuously communicates with participants while they prepare for the elicitation meeting. As soon as the requirements engineer determines that open issues which are part of the elicitation meeting's agenda have either been resolved, or have been explored and require further discussion during the elicitation meeting, the scheduling and organization activities for the elicitation meeting take place. The requirements engineer takes the lead while consulting with the customer in determining the type and subject of the meeting. Some of the meeting types, among others, can be classified to be:

- Declarative – these are meetings during which generally no requirements elicitation takes place. Here participants may present issues that are more difficult to resolve and require contacts from outside experts, or other issues or items that need to be discussed outside of elicitation
- Explorative – these are meetings during which participants will focus on identifying and discussing new issues, as well as try to identify areas of the problem domain that may impact the project. The focus of this type of meeting is on discovering and identify "new" unknowns.
- Review – these are meetings during which previously refined requirements are examined to ensure that they meet the stated needs of the customer, and are well under-

stood by all stakeholders. In general, during this meeting requirements are paraphrased such that the meaning and intent of the requirement can be confirmed.

The meeting type indicates the focus on the material to be taken, with the subject indicating the area or specific requirements or requirements sections to be examined. Besides the meeting type and subject, the requirements engineer and customer also determine the *level of detail* at which the meeting discussion should take place. We have identified three detail levels for the RGM:

1. Overview – the requirements or artifacts are discussed at the highest level, to provide the participants with an understanding of the materials.
2. Elaboration – the existing set of requirements or issues is further expanded by determining additional requirements or issues.
3. Refinement – each requirement or issue is discussed at great detail [32].

When the meeting agenda has been determined through type, subject, and level of detail, the requirements engineer, in consultation with the customer completes the rest of the meeting setup activities:

- Participants Identification – based on the meeting subject, participants that must resolve and present issues and requirements are identified and assigned responsibilities.
- Artifact Collection – the requirements engineer or an appointed meeting coordinate receives and collects all materials to be disseminated to meeting participants. Some artifacts are distributed to participants prior to the meeting with specific assignment for preparation.
- Meeting Notification – a meeting notification is send to all participants when the requirements engineer or meeting coordinator have determined (a) meeting date, time, and place, and (b) all participants required to complete assignments have indicated that they will be able to meet a predetermined deadline for completion. We have included in Appendix E a Participant Notification template (to be used for activity assignment) and a Meeting Notification template, both artifacts to be used with the RGM.

- Meeting Location – it is important to determine and reserve a meeting facility that is in an appropriate location, of size necessary for the session, and with all required equipment. Meeting location is usually determined by the location of the majority of the participants, the facility size should be such to allow for planned break-out or workshop sessions in groups, and equipment may require video displays, network connective, and conference call telephony to other offsite participants.

These meeting setup activities are part of the RGM protocol and must be completed as part of the preparation phase before the requirements elicitation sub-phase can begin; they are not optional.

*Continuous Communications*

In previous sections we have introduced many activities that revolve around the requirements engineer communicating at some level with the customer and elicitation meeting participants. At this time we want to state explicitly that continuous communications by all participants in the project is of utmost importance. We have recognized in our research that failure or improper communications leads to wrong requirements [2]. Furthermore, in the presentation of the Spiral Model [7] and Extreme Programming [5] we recognized the benefits of communication provided by each approach. Therefore, a major portion of the RGM is designed around opening communication channels, as well as controlling problematic communications (one focus of our monitoring methodology discussed in chapter 4). Continuous and open communication helps to ensure the scheduled meetings are successful. This is especially true if an individual is unable, due to other constraints, to complete necessary preparatory assignments in time. We advise to reschedule a meeting, rather than holding a meeting when a critical participant cannot attend or crucial material is not available [18]. The benefits of communication throughout the preparation phase can be found in that:

- Participants share their knowledge on open issues to help each other resolve them.
- Participants know who has been assigned similar work, or work that will be impacted through their assignment, and can communicate necessary information.
- Participants will acknowledge when deadlines cannot be met.

The preparation sub-phase is crucial in setting the stage for the successful transition to the elicitation sub-phase. It is therefore important that the requirements engineer as well as all elicitation meeting participants address and complete their assigned activities.

### 3.3.2 Elicitation sub-phase

The elicitation sub-phase is defined by the requirements elicitation meeting. Shortly after completing their assigned tasks, the participants should expect to attend an elicitation meeting during which the prepared material is presented. Therefore, becoming well prepared for the elicitation meeting is highly important, and the RGM stress preparation before each elicitation meeting (not only by naming one sub-phase "preparation"). Yet, even the best-prepared participants can still make mistakes during the elicitation meeting. Our experience and research has shown that a large number of errors are committed by both the requirements engineer and the customer during "requirements communication" sessions [23].

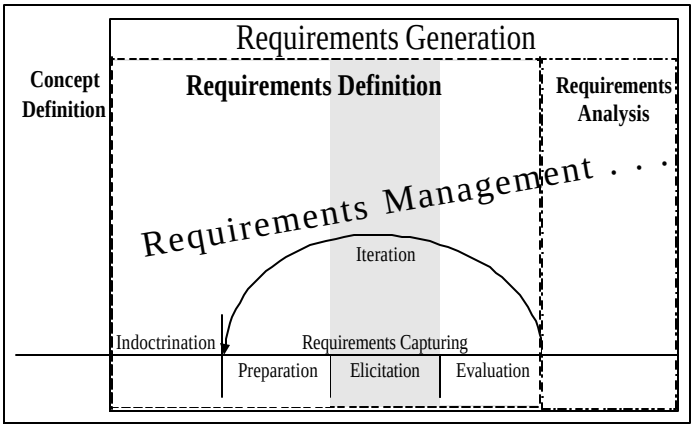



Figure 3-5**. RGF Elicitation**

The objective of the elicitation sub-phase is to correctly identify and capture requirements as communicated by the customer. This seems at first an easy task, that if not by putting pencil to paper, through audio- and video-tapings the requirements engineer should be able to precisely capture the customer's requirements. Unfortunately, the customer's spoken words all too often do not express the real intent. The requirements engineer must be aware that the customer's true intent is only implied. The problem with this implication lies in that any interpretation by the requirements engineer has a high probability of not meeting the customer's intent. Recognizing this problem, we designed the RGM to aid in accurately capturing explicitly stated requirements, as well as implied re-

quirements. This is accomplished through regular reviews after each elicitation meeting, and an interpretation of captured requirements by the requirements engineer to the customer, to assess the correct understanding of these requirements. These reviews and the interpretations therefore aid in uncovering and thereby explicitly stating previously implied requirements. The purpose of the preparation and elicitation sub-phases, as well as many of the activities within these closely mirror those found in the Fagan Inspection Process [18]. Just as Fagan expects individuals to conduct assessments prior to the inspection meeting, so does the RGM expect individuals to prepare and resolve assigned issues prior to the elicitation meeting.

The activities performed by the requirements engineer and the customer (as well as participants on behave of the customer) during the requirements elicitation meeting differ because of their respective roles. Here, the customer's responsibility is to convey all necessary and important information regarding the system to be built; precise descriptions are essential to quality and success. The requirements engineer's responsibility is to capture this information (the requirements), as well as any extraneous information the customer provides (this extraneous information in most respects presents the contextual setting for each requirement). The roles of the customer and requirements engineer during the elicitation meeting therefore differ, as do the goals each have; presenting information versus capturing information. Consequently, the approach that each sees to obtaining their goals may be different, possibly conflicting and hindering an effective quality elicitation process. However, the ultimate goal of both is the same, a quality product that meets or exceeds the customer's expectation. With this end in mind we designed the RGM to be constraining during most phases of the process, yet allow enough flexibility during the requirements elicitation meetings not to hinder the process itself.

So far we have presented guidelines and protocols for each phase; we have also designed some for the elicitation sub-phase. The weight from constraining the process through protocols shifts in this phase though to one of monitoring and correcting problems as they occur. One can say that during the elicitation sub-phase the RGM is light on protocols, yet heavy on the monitoring methodology. By designing the monitoring methodology to

allow for recognition of problems during the elicitation meeting, and providing methods to mitigate, prevent, or resolve a problem, we are providing continuous checkpoints for the meeting, but not constraining how the actual interaction and communication during the meeting must occur. In our analogy to roadways, we are sampling traffic for compliance with the rules of the road, but we do not mandate the vehicle that must be driven on our road. The details of the monitoring methodology are presented in chapter 4.

Many of the concepts we use in defining and structuring the elicitation sub-phase, and thereby the requirements elicitation meeting, are those found in well-facilitated meetings. We have selected those concepts found to address the highest number of "bad" occurrences during elicitation meeting as determined through our experience in industry as well as academic research. Furthermore, these same concepts mitigate problems that if not recognized or resolved can have a great detrimental impact on the project at a later time, possibly after product delivery. The key to presenting these concepts for the elicitation sub-phase is to raise awareness of them, and have all participants use them. This alone provides for a structure otherwise not found at regular occurring "ad hoc" meetings.

### 3.3.3 Evaluation sub-phase

The evaluation phase is the final sub-phase within the requirements capturing phase of the RGF. It is entered immediately after the completion of the most recent elicitation meeting. Within this phase the requirements engineer and the customer conduct a review of the "just captured" requirements to evaluate these in regard to a set of exit criteria as described in Appendix D. The immediate objectives of the evaluation sub-phase are described in:

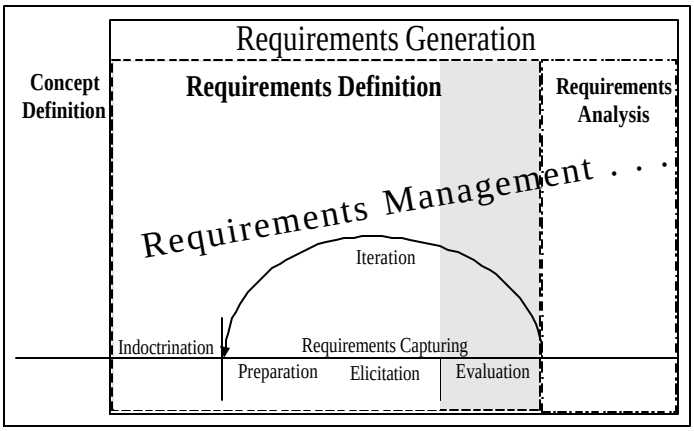



Figure 3-6**. RGF Evaluation**

(a) assessing the immediately preceding elicitation meeting to determine if the meeting objectives (inclusive of the type, subject, and level of detail) were met,
(b) examining the newly identified requirements for meeting a set of requirements attributes often only determined during requirements analysis, both as single requirements, as well as in relationship to the already existing set of requirements, and
(c) determining if an additional iteration through the requirements capturing phase is necessary, or a transition to requirements analysis can commence.

To meet these objectives, the requirements engineer has to evaluate all currently existing artifacts including: 1) the preceding meeting objectives, 2) the newly identified requirements and supporting information from the elicitation meeting, 3) the complete set of already identified requirements (taken from the requirements management tool), 4) the set of existing open issues.

*Assessing the Preceding Elicitation Meeting*

The success of an elicitation meeting is recognized in that all objectives of the meeting have been achieved. This includes the identification of all requirements related to the meeting subject, as well as gaining an understanding of issues needing further resolution. In determining this success, the requirements engineer follows a step-by-step review process prescribed by the RGM (see Appendix E for a flow of this review process). This process includes entering the newly identified requirements into a requirements management tool, together with an interpretation of each requirement. The newly entered requirements and interpretations are forwarded to the customer for examination and reinterpretation. This cycle is necessary so that the requirements engineer can determine if the interpretation of the requirements from the customer is consistent with the original assumption made about this requirement. Discrepancies must be noted and resolved either (a) directly between the requirements engineer and customer if this can be done through a short conversation, or (b) as an issue to be resolved in an upcoming requirements capturing iteration. As is apparent, the customer contributes highly to the requirements capturing process, through continuous communications with the requirements engineer. This is true during all sub-phases of the requirements capturing process.

*Examining Identified Requirements Attributes*

Aside from ensuring during requirements elicitation and the requirements review that requirements meet customer intent, the evolving requirements should be examined during each evaluation phase for meeting a set of requirements attributes. We recommend an inspection that minimally includes evaluating each single requirement for ambiguity, accuracy, testability, as well as necessity. Furthermore, the current existing set of requirements is inspected for consistency and completeness. Other requirements attributes found in sources such as the IEEE Standards [25] can be inspected for as necessary to further improve the requirements. Our recommended inspection attributes are defined by:

- ambiguity – each requirement has one and only one possible interpretation
- accuracy – each requirement precisely defines the intended functionality it is supposed to provide to the system
- testability – each requirement can be verified through an inspection or demonstration to perform its expected functionality within a reasonable amount of time
- necessity – each requirement should fulfill a necessary function that the customer asked for and intended of the system
- consistency - the existing set of requirements are consistent with each other. No conflicting terms are used, or conflicting functionality is stated
- completeness – the set of requirements describes the entire system that is to be created for the customer. All requirements are fully stated and no room left for possible different interpretations.

The activities required to establish these attributes of requirements must be tailored to the size and risk of each project. Obviously, these activities are those found as part of any standard verification and validation (V&V) process. These activities though are often time consuming when done precisely and with a lot of rigor. The RGM does not call for an indepth V&V process during the evaluation sub-phase. Rather, the requirements engineer should examine each requirement at a high level for meeting the attributes. This task could be part of the process of committing requirements to the requirements management system as well as part of the interpretation. It is important that this sub-phase is

completed quickly such that a new iteration through requirements capturing can be set up if necessary.

*Iteration or Analysis?*

At the completion of the evaluation sub-phase the requirements engineer in communication with the customer and using a set of exit criteria (Appendix D) must determine if the requirements definition 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 the answer is a clear "yes", all parties involved feel that all requirements have been captured meeting customer intent. Therefore, a transition from requirements definition (out of the evaluation sub-phase) to requirements analysis takes place. To determine this answer the set of exit criteria is examined, consisting of a checklist of items related to:

- Checking requirements attributes
- Entering all requirements in the requirements management tool
- Ensuring no open issues remaining
- Finding agreement between all parties that all requirements have been defined.

The items found on the exit criteria checklist included with the RGM are necessary items to determine "iteration or analysis" and no item shall be excluded. Yet, we encourage those using the RGM to review the exit criteria and include any additional items on the checklist as necessary for their development environment.

The structure of the RGF draws from "best practices" of other software engineering principles and approaches. We lean heavily on Fagan [18] in that the RGF requires a preparation time (parallel to the time individuals spent on inspections) prior to the elicitation meeting, as well as relying on reviews after elicitation leading to "follow-up" through further iterations. Although the RGF does not require a formal V&V approach to the reviews, we expect most requirements engineers are going to apply their V&V expertise at that time.

In this chapter we have introduced the Requirements Generation Framework and the phases which define the RGF. We have furthermore focused on the objectives and activities supporting these for each of the phases. The protocols and guidelines introduced in this chapter are a subset of those found in the Appendices. Furthermore, we discussed that the elicitation sub-phase of requirements capturing includes a monitoring methodology contingent. The following chapter focuses on this monitoring methodology of the RGM.

# Chapter 4. Monitoring Methodology

## 4 MONITORING METHODOLOGY

This chapter introduces the second component of the Requirements Generation Model (RGM), the Monitoring Methodology. The objective of the monitoring methodology is to provide a continuous quality assessment of the elicitation process during the requirements elicitation sub-phase, and provide a means for corrective action when the quality of the process is impacted. The continuous quality assessment, or monitoring, is established through specified procedures. The corrective actions are provided by methods that are invoked through the procedures when deviations from the established norm in the process are detected.

The procedures and methods described within the RGM are based on research experience and findings during a pilot study. The procedures designed for the RGM are based on the most common communication and interaction problems we found. Although we think of these problems as being common to most current elicitation processes, we do not mean to imply that the problems themselves are superficial. Rather, each problem in itself, if left undetected or unresolved, has the potential to cause expensive critical errors in the system downstream, resulting from incorrect requirements.

### 4.1 Monitoring Methodology

In chapter 3 we introduced the framework of the RGM. The framework was structured through constraining protocols, and the necessary adherence to these. These protocols are limited, though, to specifying objectives and activities integral to the RGM. They do not provide a mechanism to monitor adherence to the process of defining requirements, nor do they provide the means to correct problems in the process when detected, outside the boundaries of the RGM. These protocols work well in the design of specific tasks that either the requirements engineer or the customer must complete to stay within the constraints of the RGM. Protocols, though, do not always provide the assistance necessary to control interactions between the requirements engineer and the customer, or establish an appropriate response when there is a breakdown in the interaction. The same is true for processes that must be followed within the RGM. The monitoring methodology

is designed to further enhance the effectiveness of the RGM in those situations where protocols would be ineffective.

### 4.1.1 Mechanism of Monitoring and Correcting

Initially it may be difficult to envision how the monitoring methodology is applied during the elicitation process. Many questions come to mind, such as:

- Who does the monitoring?
- How often are things monitored?
- When are corrections made?

The answers to these are simpler in theory than in practice. All participants in an elicitation meeting are responsible for constantly monitoring the process and interactions. As soon as deviations from the process are detected, they should be corrected. That said, one may pose the practical question: "With all participants constantly monitoring the process, how are things going to get done during the meeting?" This section and the following section explain how constant monitoring integrates into the accomplishment of the tasks at hand.

First, we must state that we do not expect a requirements engineer using the RGM and the monitoring methodology for the first time on a new project to be perfect. But, this should not be of great concern, as any improvement in the requirements definition process is a step in the right direction. Second, unless a customer works with the same requirements engineering team more than once, a customer will always be aware of only the most apparent process-related difficulties.

The monitoring methodology consists of procedures that are "applied" or "used to monitor" the elicitation meeting. Procedures are stated in terminology describing a problem that could occur. For example, in this chapter we will use the following procedure for explaining the monitoring methodology and its application:

> *Recognize when the customer uses vague language in describing a system.*
> *Recognize when the customer has to "look or search for an answer."*

The context of this procedure is the recognition of the customer possibly introducing wrong or ambiguous requirements by giving vague answers to questions. Each procedure of the monitoring methodology is accompanied by at least one method that further describes the situation recognized by the procedure, and suggests corrective actions to be taken, as well as possible ways to avoid the problem in the future. The method for this procedure states the following:

> *This vagueness can be recognized by the customer answering questions using "I think so", "I believe", or other similar terminology. The requirements engineer could also lead the customer to stating requirements that are not correct through usage of leading questions that are easily answered with "Yes" or "No" instead of the customer clearly stating the requirements. It is entirely possible that the person answering these questions is not the qualified person, or is not properly prepared. Ensure any recorded vague requirements are marked as such and that they are further explored through follow-up when more information is available to the customer, or the correct person could be brought into the elicitation meeting*.

Together, the procedures and methods can detect and correct errors during the elicitation process. A list of procedures and methods developed for the RGM can be found in Appendix C. As open, honest communication is a cornerstone in defining high-quality requirements meeting customer intent, most procedures and methods are hinged on correcting communication problems during the elicitation process. Now that we have introduced an example of a procedure for monitoring the elicitation process, and a method to correct the associated problem, we focus on how the monitoring methodology is operationally applied during the process.

### 4.1.2 Operational

The operational aspects of the RGM's Monitoring Methodology fall within three categories, 1) automatic, 2) programmatic, and 3) methodical. Each of the three categories is described in this section. Procedures and methods of the monitoring methodology can

belong to one or more of these categories, depending on the problem they detect and how the problem is corrected.

*Automatic*: Automatic procedures and methods allow the requirements engineer to instantly detect when there is a deviation from the prescribed elicitation process. The term “automatic” is used loosely here, since it is contingent upon a human process rather than a mechanized tool. The requirements engineer’s familiarity with the monitoring methodology's procedures and methods directly determines his or her ability to perceive when things go wrong. Therefore, the more familiar with the monitoring methodology the requirements engineer is, the more “automatic" the detection of deviations becomes. Using our procedure example, the automatic operational aspect of recognizing vague language is realized when the requirements engineer, during the elicitation meeting, "immediately recognizes" usage of vague language, rather than finding these instances during a review of the material. This allows the engineer to address the vague language at that time, avoiding confusion or unnecessary delays. This operational aspect requires practice and experience to reach optimal effectiveness, which is not expected on the initial usage of the RGM. But customers will gain an in-depth familiarity with the procedures and methods over the course of time as they apply the RGM to multiple projects.

The monitoring methodology is designed to enhance the elicitation process and not to burden the requirements engineer and customer during that process. To this end, the procedures used for monitoring must be stated simply enough for the requirements engineer as well as the customer to understand the problem they are meant to detect; furthermore, the methods they invoke should quickly resolve the problem. Those procedures and methods thought of as automatic are related to problems commonly found in the elicitation process. A requirements engineer who has become familiar with the RGM and continuously uses it for requirements definition will become so in tune with the procedures and methods that problems are automatically detected based on previous experience. It is this previous experience and the often-used procedures and methods that make part of the operational aspect of the monitoring methodology automatic for the requirements engineer. We would like to further suggest that a requirements engineer familiar with the

RGM will not only use the RGM procedures to automatically detect problems, but will have enough foresight through these procedures to altogether avoid common problems.

*Programmatic*: Some of the procedures and methods of the monitoring methodology are programmatic. This means that they are bound to other aspects of the RGM that aid in the entire requirements definition process, and use a step-by-step approach to detect and correct process infractions. Our example procedure and method has a programmatic as well as an automatic aspect to it. The detection and prevention of vague responses to questions can be very automatic for the requirements engineer. Yet, if the requirements engineer does not immediately detect all vague or incorrect responses, the corrective method that needs to be applied is more programmatic. The programmatic method which is now appropriate states that previously introduced requirements that are based on vague responses must be removed from the current set of requirements and re-examined. This removal could possibly include a large number of requirements, depending on the dependencies they had on vague or incorrect responses. Furthermore, as all requirements are captured using the requirements management process and tool, all associated requirements will have to be inspected, step-by-step, and either removed or flagged for follow-up.

*Methodical*: The methodical operational aspect of the monitoring methodology is one that uses artifacts designed specifically to prevent deviations during the elicitation meetings, or to resolve deviations that have occurred. The example illustrating the problem of vague language does not have a direct methodical resolution. However, this operational aspect can be illustrated through the example of continuous interruptions, which is described in the next paragraph. It is important to note that not all procedures and methods fall within all three operational aspects; but each will fall within one or more.

While most of the RGM's Monitoring Methodology procedures and methods are detect/correct and automated towards communication and elicitation, some procedures and methods call on additional artifacts to be used in support of the monitoring methodology. An example here is the procedure dealing with interruptions during the elicitation proc-

ess. The procedure calls for a method to control constant interruptions during the elicitation process. These interruptions can cause various problems, as described in Scenario 4 in Appendix C. The method calls for all participants to use a "Silent Parking Lot" during the elicitation meeting to guard against unnecessary interruptions, yet to capture every possible thought that would have caused an interruption. The Silent Parking Lot is simply a piece of paper with the heading "Silent Parking Lot," which is provided to participants for the purpose of noting their thoughts, ideas, or opinions during the meeting without making unnecessary interruptions. A Silent Parking Lot template is found in Appendix E. In our pilot study, we found that simply pointing to the Silent Parking Lot when interruptions occurred caused the continuous "interrupters" to use and remember it more often.

## 4.2 Monitoring Methodology within the RGM

In this section we examine how the Monitoring Methodology fits within the RGM. We discuss the phase for which the monitoring methodology is designed, and the relationship of the monitoring methodology to the other components of the RGM.

### 4.2.1 Elicitation Phase Specific

The Monitoring Methodology is designed to aid the requirements engineer and the customer during the elicitation phase. This phase by design is very interactive, based almost exclusively on communication. Both the pilot study and previous research have shown that this phase is therefore highly susceptible to mistakes made through communication and human behavior. Most requirements engineers do not have a degree in human behavior and therefore are prone to letting these mistakes propagate into the set of elicited requirements, as well as introducing a few errors of their own. We do not want to attempt to explain or correct human behavior that introduces requirements errors. Rather, we have noticed certain events (such as interruptions) that have led to the introduction of erroneous requirements. We have taken common, often-repeated events during requirements elicitation and have designed procedures and methods around these that do not attempt to correct the behavior so much as to correct the elicited requirements. The continuous monitoring aspect can only happen during the highly interaction-oriented process. In the RGM, this process is specific to the elicitation phase. Therefore, the monitoring

methodology is designed towards the elicitation phase and the interaction within. To structure actions outside the elicitation phase, we designed both guidelines and protocols. These are not reliant on interactions, and do not monitor ongoing communication, as we next discuss.

### 4.2.2 Relationship to Other Components

In this section, we explore the relationship of the Monitoring Methodology to both the guidelines and protocols of the RGM.

*Guidelines Revisited*: As stated in chapter 3, guidelines serve in an advisory fashion; that is, the requirements engineer as well as the customer can decide which, if any, guidelines of the RGM they want to follow.

*Protocols Revisited*: Protocols are important components of the RGM. Protocols constrain the requirements engineer and the customer, thereby creating or enhancing the structure of the RGM's framework. As previously stated in chapter 3, participants must adhere to the protocols to stay within the boundaries of the RGM.

*Guidelines, Protocols, and the Monitoring Methodology*: Unlike the first two components of the RGM, the Monitoring Methodology is much more operationally oriented. The Monitoring Methodology is constantly applied during the requirements elicitation process. Guidelines or protocols are discrete components, applied or followed at specific times or instances during the requirements definition process.

A second difference makes the operational distinction between the Monitoring Methodology and guidelines and protocols even clearer. If the requirements engineer or customer decides not to use RGM guidelines, we do not expect a large negative impact on the quality of the defined requirements. Furthermore, as stated previously, it is with minimal consequence to the RGM should the guidelines be ignored. As previously stated, guidelines are additional recommendations to the activities of the RGM, but are not part of the components of constraint for the RGM. On the other hand, the require-

ments engineer and customer must use and adhere to both protocols and the monitoring methodology to stay within the pre-defined boundaries of the RGM, and to realize improvements in the defined requirements. Should the requirements engineer or customer not follow protocol for any reason, they are no longer within the constraints of the RGM. In other terms, they are no longer following the RGM. Protocols are stated in a way that they simply must be followed. There is no corrective action possible, other than to follow the protocol. If the requirements engineer or customer therefore opt not to use the RGM protocols, the beneficial results that can be obtained through following the RGM may no longer be achieved.

Protocols do not "monitor" for problems. This is the strength of the Monitoring Methodology. Not only do the procedures and methods constrain the requirements engineer and customer to work within the framework of the RGM, but also, they provide for a solution in case the Monitoring Methodology is not followed at some point in the elicitation process. Therefore, the Monitoring Methodology could be thought of as being both more constraining on the RGM structure than protocols, yet more flexible due to the possibility for error corrections. This is possible because of the two components making up the Monitoring Methodology. The procedures help in detecting problems, as well as reminding the requirements engineer and customer of necessary constraints. The methods, though, provide for corrective actions when correct procedures or processes are not followed.

In the next chapter we present a pilot study that was conducted using a subset of the components of the RGM. This pilot study enables us to test important components of the RGM and to further enhance and extend procedures and methods of the Monitoring Methodology.

# Chapter 5. Empirical Study

## 5 EMPIRICAL STUDY

The objectives of this chapter are: 1) to justify the need for an empirical study using the RGM, 2) to describe the empirical study setup, and 3) to present the collected data and provide conjectures to the study outcome.

In the previous chapters we motivate the need for a change in the way that current requirements are generated. We introduce the RGM framework that partitions and structures the requirements phase into multiple sub-phases together with guidelines and protocols. Furthermore, we discuss the monitoring methodology, which provides procedures to actively indicate when problems occur during requirements elicitation, and describes methods that can be used to correct these.

The need for an empirical study is established through the work illustrated in the previous chapters. Although common sense would have us assume that the software development process is both more efficient and more successful with structure and control of the requirements generation process such as that provided in the RGM, we cannot substantiate this assumption without an empirical study. Substantiation can not only show the correctness of our assumption, but also strengthen the need for a structured requirements generation process through the RGM. Moreover, an empirical study gives us the opportunity to observe the usage of the components within the RGM and allow for further refinement as necessary. In the next section, we focus on the setup of the empirical study to help us confirm the improvements established by the RGM.

### 5.1 Empirical Study Setup

This section describes the details of the empirical study. We focus on 1) those parts of the RGM that are deployed in the empirical study, 2) the environment in which the empirical study is conducted, 3) characteristics of the projects developed within the empirical study, 4) the people involved in the empirical study, and 5) the process used to insert the RGM into the projects developed.

### 5.1.1 Description of Empirical study

An empirical study is conducted to substantiate the benefits of the RGM. In this empirical study, we show that using the RGM over a currently existing requirements generation process does not adversely impact the requirements generation process, nor its subsequent phases. Furthermore, we believe that using the RGM improves the requirements generation process and has further positive impact on the remaining development phases through the project's deployment. Through this study, we can establish four benefits of the RGM, although many more are believed to exist:

1. By using the RGM to capture requirements as intented by the customer, we expect a positive impact on the overall project schedule, having fewer delays throughout the phases.
2. By using the RGM we expect fewer adverse impacts due to requirements changes on the overall project status throughout the development cycle.
3. By using the RGM, we expect to see a cost benefit resulting from fewer slippage days and fewer 'problems' to fix later.
4. By using the RGM we expect those involved in the project development, as well as the customer, to have a higher level of satisfaction with the project.

Changing a development environment is a difficult task at best, often due to the resistance of those working in the environment. Our empirical study is conducted in an environment greatly needing change to its "accepted" software development practices. We select a single software development department found to have difficulty in its current practices for this empirical study. Within this department, a team is selected that we train in using the RGM as their requirements definition approach. Also, within the same department, a separate group is selected as the control group, not implementing project process changes stipulated by the RGM. Furthermore, we want the changes to be minimal, but effective. Therefore we select elements from the RGM that we assume will make noticeable differences on the requirements generation process. These elements are:

- Issues/Participant Notifications (see Appendix E)
- Meeting Notifications (see Appendix E)
- Silent Parking Lot (see Appendix E)

- Limited USRL Reviews (not as extensive as the RGM requires)
- Monitoring Methodology Procedures and Methods
  - Controlled Interrupts
  - Avoiding the “Guessing Game”
  - Seeking Requirements, Not Solutions

Once we determine the groups to participate in the empirical study, and those elements of the RGM we want to study, we determine the project details. We describe these details in the next sections, followed by the presentation of the collected data and our interpretations of the empirical study.

### 5.1.2 Environment

The environment is comprised of both the physical surroundings, hardware, and operating systems utilized, as well as the atmosphere established through the organization of the study. This section details those physical elements and discusses the efforts made to avoid the introduction of bias.

*Physical Environment*

The empirical study takes place in a large financial institution. The department within which the study takes place consists of five separate groups. Four of these groups work within the same building but are physically separated. The fifth group works in a separate building. The physical environment is one of cubicles for each employee, with either a single PC running MS Windows 95, or two PCs, one with MS Windows and a second with Unix. Developers use either MS Visual Basic or C/C++ on the Unix operating system.

*Avoiding Bias*

One concern when conducting empirical studies is the introduction of bias. Bias based on the transfer of particular elements under examination in the empirical study from one project to another (or between empirical studies) compromises otherwise valid results. Introducing bias into a study is avoided by setting up an “experimental” group and a “control” group. Those elements to be studied (changes to the normal environment) are

introduced to the experimental group, while the control group continues to function the same as before. Contact between the experimental group and the control group should be non-existent or very limited during the empirical study, avoiding the possibility of transferring the study elements. Only by avoiding this transfer can we be assured that the collected data of both groups differs only due to the study elements in the experimental group.

Our empirical study employs an experimental group and a control group. The experimental group uses those previously described parts of the RGM (we refer to this group as the "RGM group") and the control group does not change their project development process (we refer to this group as the "Non-RGM group"). Between both groups we use several means to avoid bias introduction:

- Both groups work in the same department, but work in physically separate locations.
- Both groups work on the same type of applications but there is no overlap of work. Each group has a completely different set of customers.
- Only the RGM group is trained in the RGM. No mention is made outside the group as to the actual training occurring.
- Necessary artifacts are given only to the RGM group.
- We monitor both groups for the duration of the empirical study.
- Neither group is made aware that they are part of an empirical study. The collected data is part of both group's necessary job function submission requirement on projects. Therefore, competition between the groups is not introduced as part of the empirical study.

The above six items show that the possibility for introducing bias into the empirical study is very limited, virtually eliminated. Furthermore, if there is a leak of information between the groups, we would conjecture this to have a positive implication on interpreting results of the study. In view of the resulting data, we believe no bias is introduced.

### 5.1.3 Project Characteristics

This section lays out the characteristics of the empirical study projects. These characteristics include the elements that determine selection, such as project size, application type, and development and target environments. Also included are the roles of those involved in the projects.

*Project Selection*

In selecting the projects for the empirical study, it is necessary to ensure the following desirable characteristics:

- Project Time Length – must be estimated at no more than six months to completion.
- Project Size – each group must have two large projects and two small projects. Project size in the given development environment was determined by estimated project cost. A project of greater than $250K is large, and below $250K small. This cost includes the purchase of hardware as well as employee time. Size is not an indicator for the expected project length, as small projects may be developed in parallel with less than 100% human resource dedicated to a single project at a time. In this environment, it is common for small and large projects to start and finish at the same time due to this factor. Although this allows for the possibility for large and small size projects to be separated by merely dollars, if each is close to $250K, all large projects in this empirical study had a cost greatly exceeding this amount.
- Project Complexity – all projects must be related to financial processing with database access. The large projects were critical and essential to the daily operations of the business. The small projects were system functionality enhancements.
- Personnel – must have equal qualifications, experience, and length of time within their groups.

Other project characteristics are not as important to the overall selection, and are usually based on the project environment. The specific characteristics for the projects finally selected for the empirical study are as follows:

*RGM (Experimental) Group*:

- Two large projects (Projects 1 & 2).

- Two small projects (Projects 4 & 5).
- All projects are financial applications with simple data manipulation (calculations), database look-ups, and database stores.
- All projects are developed in either MS Visual Basic or C++ on Windows 95 or Unix based operating systems, respectively.
- All projects are deployed on either Windows 95 or Unix based operating system machines.
- *Roles:*
  - Large projects each have 1 project manager, 1 systems analyst, and 2 developers, as well as a single customer.
  - Small projects each have 1 project manager, 1 systems analyst, and 1 developer, as well as a single customer.

*Non-RGM (Control) Group:*

- Two large size projects initially. One project is canceled early during requirements definition. (Remaining is Project 3).
- Two small size projects (Projects 6 & 7).
- All projects are financial applications with simple data manipulation (calculations), database look-ups and database stores.
- All projects are developed in either MS Visual Basic or C++ on Windows 95 or Unix based operating systems respectively.
- All projects are deployed on either Windows 95 or Unix based operating system machines.
- *Roles:*
  - Large project has 1 project manager, 1 systems analyst, and 2 developers, as well as a single customer.
  - Small projects each have 1 project manager, 1 systems analyst, and 1 developer, as well as a single customer.

We select the Non-RGM group projects from a list of upcoming projects using only the project selection criteria stated herein. Two large projects and two small projects appro-

priate for this empirical study are chosen from a status-reporting database. The selection process for the RGM group is based on the same selection criteria, with the exception that the projects have to start after the group is trained in working with the RGM.

*Personnel*

In this section we provide an overview of the people who are involved in the projects of this empirical study. It is important for the study that all people involved in the projects have comparable skill sets in equal roles such that a meaningful data comparison can be conducted.

***Project Managers:*** There are four project managers in the empirical study (two in each group). All project managers have a minimum of ten years of experience in managing projects for various organizations, and all have been with this department for a minimum of one and a half years. Three of the project managers have an MBA degree, and one has an MIS degree.

***Systems Analysts*:** Two systems analysts (one in each group) are part of the empirical study. Both analysts have a minimum of three years of work experience as both developers and analysts. Each has a Masters Degree in Computer Science.

***Developers:*** Each group in the empirical study has four developers working on the projects. Each developer has a minimum of two years of work experience programming in both MS Visual Basic as well as Unix C/C++. Furthermore, each developer has on-the-job training in programming languages. All developers have a Bachelors Degree in Computer Science.

### 5.1.4 Empirical Study Process

For the empirical study to be successful and allow valid data to be collected, several issues have to be resolved before the study is conducted. This section focuses on the external items of the empirical study.

*Training*

The RGM group has to be trained in the selected elements of the RGM to be used during the empirical study. Only project managers and systems analysts are trained, as they alone have direct contact with the customer and are using the RGM. Training of the

group includes an overview of the RGM, as well as a discussion and reasons for the artifacts to be used for the projects. We conduct two half-day training sessions (4 hours) to educate all members of the RGM group. Furthermore, on a weekly basis we have one to two contact hours with the project manager and systems analyst to give and receive feedback on using the RGM within their projects. We do not reveal to the group that an empirical study is going to be conducted on subsequent projects to avoid introduction of bias, or other behaviors that can impact the study beyond the introduced RGM elements.

For the data collection related to this empirical study we consider training and feedback time to be a part of the regular and necessary overhead cost. Therefore, the length of time indicated for each project does not contain training time. If the RGM were to be used by a development team for a single project only, the training time and cost would have to be factored in as part of the project time. But, since the members in our empirical study continue to use the RGM on subsequent projects, the training cost becomes part of overhead, as is all other training.

*Oversight*

To ensure initial adherence to the process and to help the group with the new elements of the process, we are present at several of the requirements elicitation meetings. The project managers and systems analysts asked us to monitor their meeting progress and to make them aware when the RGM process is not followed. Only on a few occasions were interventions necessary, all of which were related to *Controlled Interrupts* and the usage of the *Silent Parking Lots.* We met with the project managers and systems analysts after the meetings to provide feedback as to their adherence to the RGM, and to receive their thoughts on elements that could be further enhanced within the RGM.

Thus far, we have determined the study elements, located an environment conducive to the study, identified qualified participants, selected appropriate projects, and trained the experimental group. The experimental group expresses commitment to follow the RGM process and to use its artifacts, and we establish methods of oversight. The study pro-

ceeds as planned, and has been concluded. The data collection and interpretation are given in the next section.

## 5.2 Data Collection, Presentation, and Interpretation

The data presented in this section are collected from a status-reporting database. Project managers are required to update project status when major milestones are reached (such as finishing the requirements phase and moving into design), and when project status changes. Project status colors are established to indicate how the project is progressing, as follows:

***Green*** – project is on target, schedule and budget are as expected. Overall the project is encountering no current risks.

***Yellow*** – project has run into a possible risk that could impact either schedule or budget. The project needs to be well-managed and the risks need to be controlled.

***Red*** – project has encountered a setback in either schedule or budget. At this time the setback needs to be evaluated and necessary project changes negotiated with the customers to resolve red status to yellow or green.

We collect further subjective data in form of observations during meetings with the customers and project managers as well as analysts. The subjective data is used in making adjustments to those parts of the RGM used in the process, as well as those not part of the empirical study, to improve the overall framework and methodology.

The objective data collected through the status reporting database serves to answer three hypotheses posed by using the RGM in the requirements generation process. We examine each hypothesis and the related data next.

### 5.2.1 Hypothesis 1

The first hypothesis states:

> *Projects using the RGM should experience less slippage in phase completion dates.*

Or stated in other terms, we expect that projects that are under the guidance of the RGM meet their projected phase completion date more often. To make this determination, we need to know the original date a phase is supposed to be completed, and the actual date

the phase is completed. Furthermore, since we designed the RGM to have a high impact on requirements generation, we examine only those phases that are immediately impacted by requirements, namely requirements definition, design, and implementation.

The data presented here is shown regarding each project for the RGM and Non-RGM groups, and the number of slippage days between the originally determined ending date for each phase and the actual date the phase ended. A negative number indicates the project ended earlier than expected, versus a positive number indicating the original project phase data was missed by the number of days indicated. A complete data view for all projects of the empirical study showing original expected phase completion dates, actual phase completion dates, as well as all calculation for the number of slippage days can be found in Appendix F. The original phase completion dates for the design and implementation phases in Appendix F reflect adjusted "reset dates." This means that if a phase finished late we adjusted the next phase's original date by the number of days the previous phase was late (i.e., if the definition phase ended two days late and the design phase had an original date of the 23$^{rd}$ we changed this date to the 25$^{th.}$. Vice versa is true if a phase was completed early).

*Data Analysis*

The data analysis for the hypothesis consists of two types of comparisons. We do a simple phase-by-phase comparison for large projects, as well as a phase-by-phase comparison for the small projects. We determine if comparable phases for the experimental group or the control group had less or more slippage dates, as well as the overall total days for each project. We use two indicators "<" and ">" to show this comparison, with "<" meaning the RGM group has less slippage for that particular phase than the control group on a same size project. The ">" is used to indicate less slippage for the Non-RGM group. We also show the actual number of days difference. Comparing all large projects in the experimental group to those in the control group, we get a total of two cross-comparison tables. For small projects we have a total of four cross comparison tables. Table 5-3 shows the relevant data and indicators for the large project comparison, and Table 5-5 shows the same for small project comparisons.

The second comparison we do is to indicate a degree of magnitude difference between all the compared large and all compared small projects in the tables from the first data analysis. We count the indicators from the first data analysis to determine a magnitude difference. For large projects, we should have two indicators per phase and total, and for the small projects we have four indicators per phase and total. The direction of the indicators gives us an understanding of the magnitude of impact the RGM has had on each project phase, as well as overall. If all indicators for a phase are "<" we set a magnitude indicator of "++" for this phase. If more than half, but not all are "<", we set that indicator to "+". If half are "<" and the other half ">" we set the magnitude indicator to "0". On the reverse side of this we use "-" and "- -" to indicate magnitudes towards the Non-RGM group. Table 5-4 shows the comparison indicator counts and the magnitude indicators for large projects, and Table 5-6 shows the data for small projects. Also included is the total number of days slippage for each comparison group.

| | **RGM Group** | | **Non-RGM Group** |
|---|---|---|---|
| Phase | Project 1 - Large | Project 2 - Large | Project 3 - Large |
| Definition | 0 | 0 | 27 |
| Design | 7 | -9 | 215 |
| Implementation | -7 | 9 | -8 |
| Total | 0 | 0 | 234 |

**Table 5-1.** Large Projects Slippage Da**ys**

| | **RGM Group** | | **Non-RGM Group** | |
|---|---|---|---|---|
| Phase | Proj. 4 - Small | Proj. 5 - Small | Proj. 6 Small | Proj. 7 - Small |
| Definition | 0 | 0 | 9 | 52 |
| Design | -3 | -1 | -2 | 1 |
| Implementation | 15 | 25 | -31 | 5 |
| Total | 12 | 24 | -24 | 58 |

**Table 5-2.** Small Projects Slippage Days

| | **RGM** | | **Non-RGM** | |
|---|---|---|---|---|
| Phase | Project 1 | Comp. | Project 3 | Diff. |
| Definition | 0 | < | 27 | -27 |
| Design | 7 | < | 215 | -208 |
| Implementation | -7 | > | -8 | 1 |
| Total Days | 0 | < | 234 | -234 |
| % Slippage | 0 | < | 330% | -330% |
| | | | | |
| | Project 2 | Comp. | Project 3 | Diff. |
| Definition | 0 | < | 27 | -27 |
| Design | -9 | < | 215 | -224 |
| Implementation | 9 | > | -8 | 17 |
| Total Days | 0 | < | 234 | -234 |
| % Slippage | 0% | < | 330% | -330% |

**Table 5-3.** Large Project Slippage Comparison

| | Comp. Dir. | Magn. | Total Differences | Avg. Difference |
|---|---|---|---|---|
| Definition | 2 < and 0 > | ++ | -54 | -27 |
| Design | 2 < and 0 > | ++ | -432 | -216 |
| Implementation | 0 < and 2 > | -- | 18 | 9 |
| Total | 2 < and 0 > | ++ | -468 | -234 |
| % Slippage | 2 < and 0 > | ++ | -659% | -330% |
| Total Magnitude | 8 < and 2 > | ++ | | |

**Table 5-4.** Large Projects Magnitude Indicators

*Interpretation: Large Projects Comparison*

The comparisons in Table 5-3 show that both RGM group projects performed better than the Non-RGM group projects in the requirements definition and design phases. The Non-RGM group finished implementation of Project 3 eight days ahead of the originally planned schedule (see Anecdotal note below for one possible reason), and therefore performed better in both comparisons over the RGM group (although in the Project 1 to Project 3 comparison only by a day). Looking at the total number of days slippage, as well as the percent slippage, the two RGM group projects (Project 1 and 2) finished the combined definition, design, and implementation phases on the target date, therefore having zero percent slippage. The Non-RGM project did not fare as well. Due to the large setback in the design phase, the project came in 234 days late, or 330% slippage over the initially planned schedule. Table 5-4 summarizes the project comparisons and indicates if there is a substantial difference when using the RGM over a traditional development process. With exception of the implementation phase comparison, all other phases indicate, along with the totals, that the group using the RGM for their development process performed better than the Non-RGM group.

The data showing that the RGM group large projects did not slip in schedule, and that the Non-RGM group project slipped by a very large percentage, enables us to come to a conjecture on Hypothesis 1 for large projects. We conjecture that for large projects, Hypothesis 1 holds true. That is, for large projects of comparable size and complexity, with a project team having similar experiences and knowledge, the RGM does not negatively impact the projects. Rather, projects using the RGM should experience less slippage due to customer intents being correctly defined early, and thereby, requiring less rework in later phases.

**Anecdotal note:** There is no definitive data to determine why the Non-RGM Project 3 completed implementation ahead of schedule. We notice that often developers for the Non-RGM group start implementation much before requirements or design documentation is completed. Implementation is based on “knowledge” about what the overall project’s goal seems to be. We also notice that the “base” code that stems from this early

work is later modified and changed to meet the "actual" requirements or, less often, the actual design is modified to meet the requirements. Without proof of data, but based on the above observation, we would like to further conjecture, that for this study "The operational correctness of software developed using the RGM is substantially better than software developed without the benefit of the RGM."

*Interpretation: Small Projects Comparison*

Table 5-5 shows the possible four project comparisons between the two RGM group projects and the two Non-RGM group projects. Here, the RGM group experienced no slippage for the requirements definition phase and ended the design phase for both projects a few days earlier than expected. The RGM group did experience some slippage in both projects during implementation, twelve and twenty-four days of total slippage (13% and 26% slippage) for Projects 4 and 5, respectively. Yet, when compared to the Non-RGM group the RGM group still experienced a benefit. The Non-RGM group experienced slippage for both projects during definition. It did do better in the design phase, with Project 6 finishing two days early and Project 7 just one day late. When comparing the implementation days for the RGM and Non-RGM group small projects it would seem that similar to the large projects, the small Non-RGM group projects overall fared better than those of the RGM group. This is found in the fact that Project 6 finishing about a

| | **RGM** | | **Non-RGM** | |
|---|---|---|---|---|
| Phase | Project 4 | Comp. | Project 6 | Diff. |
| Definition | 0 | < | 9 | -9 |
| Design | -3 | < | -2 | -1 |
| Implementation | 15 | > | -31 | 46 |
| Total Days | 12 | > | -24 | 36 |
| % Slippage | 13% | > | -12% | 25% |
| | | | | |
| | Project 4 | Comp. | Project 7 | Diff. |
| Definition | 0 | < | 52 | -52 |
| Design | -3 | < | 1 | -4 |
| Implementation | 15 | > | 5 | 10 |
| Total Days | 12 | < | 58 | -46 |
| % Slippage | 13% | < | 123% | -110% |
| | | | | |
| | Project 5 | Comp. | Project 6 | Diff. |
| Definition | 0 | < | 9 | -9 |
| Design | -1 | > | -2 | 1 |
| Implementation | 25 | > | -31 | 56 |
| Total Days | 24 | > | -24 | 48 |
| % Slippage | 26% | > | -12% | 38% |
| | | | | |
| | Project 5 | Comp. | Project 7 | Diff. |
| Definition | 0 | < | 52 | -52 |
| Design | -1 | < | 1 | -2 |
| Implementation | 25 | > | 5 | 20 |
| Total Days | 24 | < | 58 | -34 |
| % Slippage | 26% | < | 123% | -97% |

**Table 5-5.** Small Project Slippage Comparison

| | Comp. Dir. | Magn. | Total Differences | Avg. Difference |
|---|---|---|---|---|
| Definition | 4 < and 0 > | ++ | -122 | -30.5 |
| Design | 3 < and 1 > | + | -6 | -1.5 |
| Implementation | 0 < and 4 > | -- | 132 | 33 |
| Total | 2 < and 2 > | 0 | 4 | 1 |
| % Slippage | 2 < and 2 > | 0 | -144% | -36% |
| Total Magnitude | 11 < and 9 > | + | | |

**Table 5-6.** Small Projects Magnitude Indicators

month early, and Project 7 just having five days slippage. Our reasoning for this is, again without data proof, that Non-RGM developers started implementation early during the project before definition and design were complete. To draw a better conclusion from this data we next examine the magnitude indicators.

Table 5-6 gives a summation of the small project comparison with magnitude indication. The magnitude indication shows that the RGM group only experienced a greater benefit than the Non-RGM group during definition and a smaller benefit during design. During the implementation phase the RGM group did worse than the Non-RGM group and overall there does not seem to be a benefit for small projects using the RGM. Yet, when the actual numbers for slippage percentage are compared, the RGM group experienced 36% less slippage on average over the projects than the Non-RGM group. This indicates that even for small projects there is a benefit of using the RGM for the development process. Therefore, we conjecture that for small projects Hypothesis 1 holds true. That is projects employing the RGM exhibit less variability in their ability to meet originally specified completion dates.

*Additional Observations*

In terms of slippage percentage the data in the previous tables suggests that with increased project size there is an increased benefit of using the RGM. Additionally, we expected that the RGM would contribute some overhead through increased communication

to the definition phase. For the selected projects, however, there is no indication that this is indeed the case. Rather, it appears that the RGM, as intended by design, added no extra time to the requirements definition phase of the RGM group projects.

We have also previously speculated why for the Non-RGM group, the implementation phases seemed to progress better than for the RGM group. In particular, data from Project 6 of the Non-RGM group seems to indicate that projects without the RGM can be developed on a schedule no worse than those using the RGM. Here, as a result of beginning implementation before design is completed, an operational correctness examination may reveal conjectured differences. We reassert that the RGM process results in a better end product than that of a Non-RGM process.

In conclusion we conjecture that Hypothesis 1 holds true for all size projects. This can be based on the following observations:

- Groups using the RGM are enabled to meet their original project completion dates more often or are much closer to the date.
- The data in the above tables suggest that with the increase in project size, there is a corresponding increase in the benefit of using the RGM.
- Before the experiment began we expected that the structured activities and constraints enforced by the RGM would add additional communication overhead (and possibly delays) to the earlier phases of the development process. For the projects following the RGM there is no indication that such was the case.

To summarize, the observations, conjectures and interpretations of the data provided above *suggest* that, from an overall perspective, the projects employing the RGM fair better than the other projects.

### 5.2.2 Hypothesis 2

The second hypothesis states:

> *Projects using the RGM should exhibit less adverse impact stemming from late discovery of requirements.*

Our experience [3] and that of others [9, 35, 42] indicate that the late discovery of requirements can have a detrimental impact on project cost and schedule. We believe that by using the RGM, all customer requirements and intentions are identified early and, therefore, the project experiences less requirements change in later phases than a project progressing without the benefit of the RGM. Our experience has shown that some scope and requirements changes are to be expected. These changes, similar to requirements errors, when discovered early, can be included in the project (assuming necessary scope, budget, and time allow for it) for a fraction of the cost it would take to include them during the later phases [8]. Furthermore, using the RGM, the number of changes due to initially undiscovered requirements should also be minimized.

To discover changes stemming from late requirements discovery, we again use the project status database. Here it allows one to note project status using three project characteristics: schedule, budget, and overall project. To draw conclusions for Hypothesis 2, we count the number of status changes stemming from requirement problems for each project characteristic, independent of the phase of the project. Project status changes are indicated by three possible "bad" (from green to red) shifts and three "good" (red to green) shifts. These six possible shifts are: 1) (G)reen to (Y)ellow, 2) Green to (R)ed, 3) Yellow to Red, 4) Red to Yellow, 5) Red to Green, and 6) Yellow to Green. The tables below show the number of times status changed for each project in the experimental and control groups.

| | **RGM Group** | | | | | | **Non-RGM Group** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Project 1 – Large | | | Project 2 - Large | | | Project 3 - Large | | |
| | Sched. | Budg. | Over. | Sched. | Budg. | Over. | Adj. Sched. | Adj. Budg. | Adj. Over |
| G->Y | 0 | 0 | 0 | 0 | 0 | 0 | -2 | -2 | -2 |
| G->R | 0 | 0 | 0 | 0 | 0 | 0 | -2 | 0 | 0 |
| Y->R | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 |
| R->Y | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| R->G | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 |
| Y->G | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 1 |
| Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

**Table 5-7.** Large Projects Status Change

| | RGM Group | | | | | | Non-RGM Group | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Project 4 - Small | | | Project 5 - Small | | | Project 6 - Small | | | Project 7 - Small | | |
| | Sched. | Budg. | Over | Sched. | Budg. | Over | Adj. Sched | Adj. Budg | Adj. Over | Adj. Sched | Adj. Budg | Adj. Over |
| G->Y | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | -3 | -1 | -2 |
| G->R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Y->R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | -1 |
| R-Y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R->G | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 2 |
| Y->G | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | -1 |

**Table 5-8**. Small Projects Status Change

Legend for Table 5-7 and Table 5-8:

*Adj. Sch.* - Adjusted Schedule calculation (see G*)

*Adj. Bud.* - Adjusted Budget calculation (see G*)

*Adj. Over.* - Adjusted Overall calculation (see G*)

*Indicators:* (G)reen - No current impacts on project, (Y)ellow - Impacts possible, must be managed, (R)ed - Impact occurred on project, must be resolved

Indicator Change: Each occurrence of a status changes based on requirements from the Green direction towards Red (G->Y,G->R, Y->R) is indicated by a (negative) -1 (totals for each change over project development time are shown). In the direction from Red to Green each occurrence is indicated by a (positive) 1.

*Value*: Indicates the overall total of the status changes. Projects do not necessarily start or end in Green. Therefore, the value is not necessarily zero.

G* - In the normal calculation, status changes from G->Y and G->R are assigned a value of -1 (or 1 going in the opposite direction). In the adjusted columns a change from G->R receives a value if -2, equivalent to a change from G->Y and subsequent change from Y->R (or 2 going from R->G). This adds an indicator of magnitude to the data analysis.

*Interpretation: Status Change Data*

The benefits of using the RGM for a development process over a "traditional" development process become apparent in the data presented in Table 5-7 and Table 5-8. The two large projects and the two small projects of the RGM group experienced no status changes due to late discovery of requirements in the three categories (schedule, budget, and overall) during the project duration. On the other hand, the single large project and the two small projects for the Non-RGM group all experienced some project status changes. Here, Project 6 has a single change towards yellow and a change back to green. (As we see in the interpretation for Hypothesis 1, this project performs better than expected overall.) The other two Non-RGM projects shift to yellow several times, and each also shift to red. The difference between the number of status changes from the RGM

group projects and the Non-RGM group projects allows us to conjecture that, for all projects in the empirical study, Hypothesis 2 holds true.

To summarize, the data shown in Table 5-7 and Table 5-8 indicate that the projects employing the RGM had no adverse impact stemming from late discovery of requirements. Conversely, the data also shows that the Non-RGM projects, i.e. those that continued with the established development process, did incur undesirable schedule and budget changes due to late discovery of requirements. We assert, therefore, that projects employing the RGM can realize the earlier (and more complete) discovery of requirements, and thereby, minimize the adverse impact stemming from the later discovery of requirements.

**Anecdotal Note:** In an attempt to show a magnitude of change in status, each change towards red is given the value "-1," and each change towards green a value of "1." Furthermore, in both Table 5-7 and Table 5-8 an "Adjusted" column replaces the regular three columns. In the adjusted column, a step from green to red is given a value of "-2," and from red to green a value of "2" indicated a "two" level status change. If all projects started with a status of "green" and ended with a status of "green," the sum of these values is zero. Yet, Non-RGM Projects 3 and 7 do not have a sum of values of zero for all categories. This is an indication that either the project started in a status different from green (which is not uncommon), or a project ended in a status other than green. This is the case for Project 7, which ended in a red status. Post-project discussion with the project manager reveals that the project has multiple setbacks due to misunderstood requirements. The project is completed almost two months past the original promised date. At delivery time, the customer did not accept the product and had by that time made the decision to seek a similar product from a separate development team. As shown in the data presented next, this frustration with the project is apparent.

### 5.2.3 Hypothesis 3

Our third hypothesis states:

> *Projects using the RGM result in satisfaction of the project manager and business customer a greater percentage of time throughout the project to closure.*

Unlike the previous two hypotheses, which focus on objective project data, the data for this hypothesis are more subjective in nature. This data tracks the level of frustration as directly expressed by the project manager and customer involved in the project. The project manager records this satisfaction or frustration with the project in the project status database when the project status changes or milestones are reached. We compare the satisfaction indicators "yes" or "no" to determine the percentage of satisfaction versus frustration over the project development time for the Project Manager (PM) and the Business Customer (BC). Table 5-9 and Table 5-10 show the data for satisfaction for the RGM group and the Non-RGM group. The total number of times the PM and BC are satisfied (yes) and frustrated (no) are shown, as well as a percentage of yes/no time for the PM and BC (PM % and BC %) at each level.

*Interpretation: Customer Satisfaction Data*

Confirming what is also implied by the data presented for Hypothesis 2, Table 5-9 and Table 5-10 show that the PM and the BC are satisfied with the RGM projects 100% of the time. This is not surprising since the four projects meet or are close to their expected delivery dates (as indicated by data from Table 5-1) and there are no status changes in the projects. Overall, this shows that the projects using the RGM succeed and meet customer expectations. The satisfaction data for the Non-RGM group in Table 5-9 and Table 5-10 also indicate a similar picture to that drawn by its data for Hypothesis 2. Again, Project 6 fares better than the other two projects (at 100% satisfaction of both PM and BC), but for the other two projects the BC is frustrated at least 1/3 of the time. Even a stronger level of frustration is indicated by the PMs of these two projects, who report being frustrated almost 50% of the time or more with the project progress. Moreover, for Project 7 the PM indicates an even higher level of frustration than the BC.

| | RGM | | | | | | | | Non-RGM | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Project 1 - Large | | | | Project 2 - Large | | | | Project 3 - Large | | | |
| | PM | PM % | BC | BC % | PM | PM % | BC | BC % | PM | PM % | BC | BC % |
| Yes | 5 | 100% | 5 | 100% | 5 | 100% | 5 | 100% | 8 | 57% | 9 | 64% |
| No | 0 | 0% | 0 | 0% | 0 | 0% | 0 | 0% | 6 | 43% | 5 | 36% |
| Total | 5 | | 5 | | 5 | | 5 | | 14 | | 14 | |

**Table 5-9.** Large Projects Satisfaction

| | RGM Group | | | | | | | | Non-RGM Group | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Project 4 - Small | | | | Project 5 - Small | | | | Project 6 - Small | | | | Project 7 - Small | | | |
| | PM | PM % | BC | BC % | PM | PM % | BC | BC % | PM | PM % | BC | BC % | PM | PM % | BC | BC % |
| Yes | 6 | 100 | 6 | 100 | 3 | 100 | 3 | 100 | 9 | 100 | 9 | 100 | 4 | 44% | 6 | 67% |
| No | 0 | 0% | 0 | 0% | 0 | 0% | 0 | 0% | 0 | 0% | 0 | 0% | 5 | 56% | 3 | 33% |
| Total | 6 | | 6 | | 3 | | 3 | | 9 | | 9 | | 9 | | 9 | |

**Table 5-10.** Small Projects Satisfaction

Based on the data in Table 5-9 and Table 5-10, we conjecture that Hypothesis 3 holds true. That is, project managers and business customers associates with projects employing the RGM tend to experience a higher level of satisfaction throughout the development effort than do their counterparts associates with the Non-RGM project development efforts.

**Anecdotal Note:** During post-project reviews conducted on Project 2 and 4 (two RGM group projects) the business customer and other stakeholders involved in the project express their highest levels of satisfaction with the project progress. Furthermore, they expressed opinions that their needs were always considered and that every effort was made by the development team to meet or exceed their expectations. The business customer remains favorably disposed towards using the same development team for future projects.

Regarding the RGM, the *Silent Parking Lot* artifact is judged by those involved as one of the best tools used during requirements elicitation and other meetings. The feeling of "not forgetting" necessary items, as well as "being heard without speaking" is articulated as a major benefit of using the *Silent Parking Lots.*

### 5.3 Conclusion

The data presented for the RGM group projects in comparison to the Non-RGM group projects leads to the conclusion that the three hypothesis regarding benefits of projects using the RGM are true. The benefits of employing the RGM in a development effort are:

1. Reduced slippage in milestone completion dates,
2. Fewer late discovery of requirements, and therefore, less adverse impact stemming from such discoveries, and
3. Substantial customer and management satisfaction levels.

We reiterate that the elements of the RGM introduced in this empirical study are only a subset of the entire RGM, yet very effective. We conjecture that including those elements not used by the study as part of an operational RGM can only further strengthen and structure the all-too-often ad hoc requirements definition processes. For example, the

groups within which this study is conducted did not have access to a Requirements Management tool. Requirements Management and a supporting tool is an integral part of the RGM, allowing efficient and easy refinement and changes to the requirements. Moreover, Requirements Management enables the requirements engineer and the customer to easily track and identify changes to requirements that have an overall impact on the entire project. Structuring, monitoring, and controlling the requirements definition process are the goals of the RGM. We do this through the established guidelines and protocols that work within the framework introduced in Chapter 3, as well as the monitoring methodology as discussed in Chapter 4. One major contributor to the improvement in the process, which is apparent to all participants of the RGM group in this study, is the increased level of awareness of the customer's needs, due to the overall increase in effective communication that is taking place.

# Chapter 6. Summary and Future Work

## 6 SUMMARY AND FUTURE WORK

In the previous chapters we have introduced an evolution of software development, from early "chaos" through current efforts in requirements engineering. We motivate the need for a structured requirements definition process, and presented the RGM with all its components as a solution approach. Finally, we present an empirical study giving validity to the RGM as used in an industrial setting. We shortly summarize this body of work and present future opportunity for work related to the RGM.

### 6.1 Summary

This work was motivated by the findings through experience and many research projects revealing the shortcomings in the "common" requirements elicitation settings. We found that a lot of work had been done in the area of software testing (ensuring the software that has been implemented performed reliably), as well as in the area of requirements analysis, ensuring the requirements document is an accurate representation of the elicited requirements. Yet, we found only a few attempts at improving the actual requirements elicitation process, none of which are concerned with "meeting customer intent." The results of these shortcomings are systems that have "good requirements" documentation, and software that performs as described in this documentation, but not systems that performed to the true requirements as envisioned by the customer.

The RGM was created to make the requirements elicitation process customer-centric. The objectives are to (a) make the process easy for the customer to understand and participate in, (b) make it easy to integrate the process into an existing development environment (through extensions), (c) minimally increase the amount of time spent during requirements elicitation, while aiding in possibly decreasing the overall amount of time spend in the development process, and (d) create the process such that it constrains but does not constrict the requirements elicitation effort.

In designing the RGM we evaluated existing software and requirements engineering processes. From these we adapted methods that worked well, such as splitting the require-

ments definition into smaller, more defined phases, and establishing meetings that call on the many benefits of Fagan's inspection process [18]. We also recognized that many of the existing processes through the software development lifecycle are narrowly focused in their goals and objectives, yet there is no overarching framework that ties these individually beneficial efforts together. We therefore created the RGF to provide an umbrella structure over the defined phases and their integral guidelines and protocols. This effort results in the Requirements Generation Model that consists of a framework and the monitoring methodology designed to improve the requirements definition process. The improvement is noticed in projects having less requirements changes and higher customer satisfaction levels as is demonstrated by our empirical study.

## 6.2 Contributions

The benefit the RGM provides to computer science, and in particular to software and requirements engineering is realized in improved requirements meeting customer intent. This is achieved through the RGM which structures, redefines and refines the activities and processes by which requirements are elicited, recorded and evaluated. Through that structure and refinement, the RGM defines an environment within which a more effective process resides for capturing requirements reflecting customer needs and intent. The major components of that model are:

- a framework reflecting a refinement (or decomposition) of the requirements generation phase into four mini-phases – each mini-phase has its own unique set of objectives and supporting activities,
- guidelines and protocols designed to structure the interaction process and to guide the customer and requirements engineer in their discussion, and
- a monitoring methodology consisting of procedures and methods that continually monitor the requirements generation process, look for deviations from prescribed norms, and suggests actions when such deviations are identified.

The benefits of the RGM can be realized by its contribution to a software development process that more precisely meets customer intent, and delivers software on time and

within budget as is demonstrated in our pilot study. In particular, the study provides data that reinforces our contention that the RGM helps

- reduce schedule slippage,
- through the early and more complete identification of customer requirements, minimize the adverse impact stemming from the late discovery of requirements, and finally,
- promote higher-levels of customer and project manager satisfaction.

The RGM can continue to evolve as new refinements to conventional activities are applied, as well as new ones introduced.

## 6.3 Future Work

The RGM is designed to structure and constrain the requirements definition process as embarked upon by a requirements engineer and customer. Both protocols and a monitoring methodology create constraints. Yet, we did not want to constrict the requirements engineer and customer in their freedom to create and communication, something that is often very necessary in the difficult task of developing software. The following three paragraphs highlight some of the possibilities for future development efforts regarding the RGM.

*Integrating RGM with other Software Development Methodologies*

We purposely avoided tying the RGM to any one large software engineering methodology (such as Waterfall [34] or Spiral model [6]). Our intention is to enable the RGM to be used within any software engineer approach, either substituting or supplementing an existing requirements definition process. Exactly how the RGM most effectively integrates with these and other development methodologies and models is ripe for explanation.

*Expanding protocols, guidelines, procedures and methods*

The RGM was designed with extension in mind. The protocols and monitoring methodology provided with the RGM must be used as stated (otherwise not all envisioned bene-

fits can be realized), but for any given environment necessary additional protocols as well as procedures and methods can be included with the RGM. This allows a development team to customize the RGM through extension, without deleting the existing components of the RGM. Additional research is needed to identify protocols, procedures and methods that assume the characteristic qualities of other development approach, e.g. OO.

*Environment for Requirements Generation*

The RGM contains a requirements management (RM) contingent. The RM tool used with any development effort applying the RGM is decided on by the requirements engineer. A large choice of RM tools is available on the market fitting every budget size. Unfortunately, unlike the wide availability of RM tools, there is a lack of tools that directly support the requirements elicitation process. At the time of the writing of this work, only a single company has been exploring the development of an elicitation-supporting tool that directs the requirements engineer through a predetermined workflow. This tool will furthermore integrate with an RM tool marketed by the same company. Due to a buy-out of this company, the future of this requirements elicitation tool is as of yet unknown. We envision and support the development of a requirements elicitation tool that uses the RGM as the guiding workflow and process for requirements elicitation.

## References


[1] M. W. Alford, "Management of Requirements Development Using SREM Technology," Huntsville, AL 1977.

[2] J. D. Arthur and M. K. Gröner, "Lecture Notes, Independent Verification and Validation Course." Virginia Tech, 1995.

[3] J. D. Arthur, M. K. Gröner, K. J. Hayhurst, and C. M. Holloway, "Evaluating the Effectiveness of Independent Verification and Validation," *IEEE Computer*, vol. 32, pp. 79-83, 1999.

[4] C. Ashworth, "S tructured Analysis and Design Method (SSADM)," in *Information and Software Technology*, vol. 30, 1988, pp. 153-163.

[5] K. Beck, *Extreme Programming Explained: Embrace Change*. New York, NY: Addison-Wesley, 1999.

[6] B. Boehm, "A Spiral Model for Software Development and Enhancement," *Computer*, vol. 21, pp. 61-72, 1988.

[7] B. Boehm, "Using the WINWIN Spiral Model: A Case Study," *Computer*, vol. 31, pp. 33-44, 1998.

[8] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice-Hall Inc., 1981.

[9] B. W. Boehm, "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software*, vol. 1, pp. 75-88, 1984.

[10] E. Bravo, "The hazards of leaving out the users," in *Participatory Design: Principles and Practices*, D. Schuler and A. Namioka, Eds. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc. Publishers, 1993, pp. 3-11.

[11] E. Carmel, R. D. Whitaker, and J. F. George, "PD and Joint Application Design: A Transatlantic Comparison," *Communications of the ACM*, vol. 36, pp. 40-47, 1993.

[12] D. I. A. Cohen, *Introduction to Computer Theory*, 2nd. ed. New York, NY: John Wiley & Sons, Inc., 1996.

[13] A. M. Davis, *Software Requirements: Objects, Functions, and States*, Revision ed. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1993.

[14] A. M. Davis and D. A. Leffingwell, "Making Requirements Management Work for You," Rational Software Corporation 1999.

[15] S. E. Directorate, "Software Engineering Evalutation System (SEES)," vol. III. U.S. Army Missile Command; Redstone Arsenal, Alabama: NASA Headquarters, 1994.

[16] R. G. Ebenau and S. H. Strauss, *Software Inspection Process*. New York, NY: McGraw Hill, 1993.

[17] P. Ehn, "On Participation and Skill," in *Participatory Design: Principles and Practices*, D. Schuler and A. Namioka, Eds. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc. Publishers, 1993, pp. 41-77.

[18] M. E. Fagan, "Design and Code Inspection to Reduce Errors In Program Development," *IBM Systems Journal*, vol. 15, 1976.

[19] M. E. Fagan, "Fagan Associates," 2001.

[20] A. Gibbons, *Algorithmic Graph Theory*. Cambridge, MA: Cambridge University Press, 1985.

[21] T. Gilb and D. Graham, *Software Inspection*. New York, NY: Addison-Wesley, 1993.

[22] J. Greenbaum, "A Design of One's Own: Towards Participatory Design in the United States," in *Participatory Design*, D. Schuler and A. Namioka, Eds. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc. Publishers, 1993, pp. 27-37.

[23] M. K. Gröner and J. D. Arthur, "An Operational Model Supporting The Generation of Requirements That Capture Customer Intent," presented at 17th Annual Pacific Northwest Software Quality Conference, Portland, OR, 1999.

[24] S. Group, *CHAOS Chronicles II*. West Yarmouth, MA, 2001.

[25] IEEE, *IEEE Standards Software Engineering Collection*. New York, NY: IEEE, 1999.

[26] S. E. Institute, "Capability Maturity Model Integration," vol. 2002, 1.1 ed: Carnegie Mellon University, 2002.

[27] R. O. Lewis, *Independent verification and validation: a life cycle engineering process for quality software*. New York, NY: John Wiley & Sons, Inc., 1992.

[28] R. M. Linger and H. D. Mills, "A Case Study in Cleanroom Software Engineering: The IBM COBOL Structuring Facility," presented at COMPSAC '88, Chicago, 1988.

[29] P. Mambrey, R. Oppermann, and A. Tepper, *Computer und Partizipation: Ergebnisse zu Gestaltungs- und Handlungspotentialen*. Opladen: Westdeutscher Verlag GmbH, 1986.

[30] G. J. Meyers, *Software Reliability: Principles and Practives*. New York, NY: John Wiley & Sons, 1976.

[31] H. D. Mills, M. Dyer, and R. M. Linger, "Cleanroom Software Engineering," *IEEE Software*, vol. 4, pp. 19-24, 1987.

[32] R. E. Nance, "The Conical Methodology: A framework for simulation model development," presented at Methodology and Validation, Orlando, Florida, 1987.

[33] S. L. Pfleeger, *Software Engineering: The Production of Quality Software*. New York, NY: Macmillan Publishing Company, 1987.

[34] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 5th ed. New York, NY: McGraw-Hill, 2001.

[35] J. Radatz, "Analysis of IV&V Data," Rome Air Development Center, Griffiss AFB, N.Y. RADC-TR-81-147, June 1981 1981.

[36] S. Robertson and J. Robertson, *Mastering the Requirements Process*. Harlow, England: Addison-Wesley, 1999.

[37] D. Ross, "Applications and Extensions of SADT," *IEEE Computer*, vol. 18, pp. 25-35, 1984.

[38] W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," presented at WESCON, 1970.

[39] R. W. Sebesta, *Concepts of Programming Languages*, 3rd ed. Menlo Park, CA: Addison-Wesley, 1996.

[40] I. Sommerville, *Software Engineering*, 5th ed. Reading, Mass.: Addison-Wesley Publishing Co., 1996.

[41] International Organization for Standardization, "ISO9000," vol. 2002: ISO, 2000.

[42] D. R. Wallace and R. U. Fujii, "Software Verification and Validation: An Overview," *IEEE Computer*, vol. 6, pp. 10-17, 1989.

[43] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt, "Automated Quality Analysis of Natural Language Requirements Specifications," presented at 14th Annual Pacific Northwest Software Quality Conference, Portland, 1996.

[44] N. Wirth, *Algorithms and Data Structures*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1985.

## Appendix A. RGM Protocols

*Protocols* – are rules that a) establish boundaries through pre-defined constraints, and b) impose operational, goal-oriented actions through mandates. Unlike guidelines, protocols do establish the structure imposed on the framework and therefore must be adhered to. Protocols are established to ensure that participants in the requirements definition process "operate" within well-defined boundaries. They are effective only if followed. When violated, it is the responsibility of the participants to rectify that "violation."

The following is a list of protocols designed for the RGM. The protocols are grouped by the RGM phase during which they are to be applied. It is necessary for a requirements definition process using the RGM to follow all protocols. Furthermore, a requirements engineer may establish additional protocols, while being aware not to "overburden" the constraints.

All Phases:

- Meetings must start on time
- Participants must arrive at meetings on time
- Participants must be prepared for meetings, and must understand the materials they are to present
- Participants must at all times be honest about the materials presented as well as feel comfortable to state their own opinions
- Meetings must be limited to two-hour sessions. If multiple sessions are needed, they must allow for 20-30 minute breaks between them

Indoctrination Phase:

- Stakeholders (all current) must be identified, and a list of stakeholders must be modified to include newly identified ones
- Responsibilities (tasks assigned to each participant) must be established for the project

- The requirements engineer must receive an “overview” of the customer’s current domain and environment
- A Requirements Management tool must be set up for the project. “Uncommon” data elements must be identified

Requirements Capturing Phase:

- Meeting notifications (see Appendix E for template) must be send to all participants no later than one week prior to the meeting
- Responsibilities and tasks must be assigned to individuals prior to the initial meeting, adjusted when needed, and always communicated to all meeting participants
- The type of meeting must be determined and announced such that all participants can prepare appropriately (meetings types can be declarative, explorative, elicitation, reviews, or others as needed)
- Meeting detail levels must be determined and announced (overview, elaboration, refinement)

Preparation Sub-Phase:

- Material prepared by participants for the meeting must be submitted to the meeting coordinator by the requested due date
- Discussion material must be organized and distributed by the meeting coordinator together with the meeting notification to all participants
- Meeting facilities must be determined, inspected for appropriate equipment (overheads, video, network connections) and announced
- Participants with schedule conflicts must notify the meeting coordinator in a timely manner
    1. related to time to prepare materials
    2. related to the scheduled meeting time/place

Elicitation Sub-Phase:

- Meetings must start on time

- Meetings must stay on the subject and at the declared detail level
- Elicitation meetings should be no longer than two hours to uphold their effectiveness
- Silent Parking Lots (see Appendix E for template) are to be used by all meeting participants to record “pop-up” thoughts, unrelated but relevant issues, and other items that contribute to the project.
- Beepers and cell phones must be in vibrate or off position.
- Time limits must be set and held for each elicitation meeting.
- Participant responsibilities must be well defined and executed during the elicitation meeting

Evaluation Sub-Phase:

- Conduct Requirements Review on a) newly discovered requirements, b) changed requirements, and c) resolved issues.
- Enter newly captured requirements into Requirements Management Tool and update existing requirements with new information
- Determine if exit criteria (see Appendix D for checklist) are met
- Select new issues, participants for next iteration
- Send out Issues/Participant Notification (see Appendix E for template) to all project members

## Appendix B. RGM Guidelines

*Guidelines* – are suggestions or recommendations that offer support by serving in an advisory capacity. Guidelines are not constraining, but rather refine the RGM. Non-adherence does not invalidate the structured process of the RGM.

The following is the list of guidelines designed for the RGM shown by the RGM phase during which they can be applied. A requirements engineer may establish additional guidelines, keeping in mind not to "overburden" the suggested tasks.

All Phases:

- Communication is important – make all questions/problems/issues known to other participants
- Mistakes can happen – be alert and avoid/identify/resolve mistakes
- No question is a stupid question – ask the unknown
- Feel comfortable in voicing your opinion – others may share it
- Use best practices – draw upon others' experiences
- Always show respect to those you are working with

Indoctrination Phase:

- There may be a learning curve – be patient and understanding
- Keep your mind open to new knowledge
- Do not judge, or pre-determine the solution

Preparation Sub-Phase:

- Know your abilities. Involve others (experts), share credit
- Solve a problem in a team if possible
- Gather different points-of-view as they may provide additional and complementary understanding

Elicitation Sub-Phase:

- Respect the opinion of others (use appropriate techniques to control a meeting)
- Set out to end meeting 10 minutes early to allow for necessary side discussion, questions, and other “finish-up” discussions not related to meeting subject
- Find the “best” requirement, even if it is not your own

Evaluation Sub-Phase:

- It may be necessary to iterate one more time, even if you understand the entire problem but others don’t

## Appendix C. Sample RGM Procedures and Methods

**Scenario 1:** During a requirements elicitation meeting implementation alternatives are being suggested.

*Problem:* Although alternative solution approaches are an important part of the solution process, they have no place during requirements elicitation. Implementation suggestions must not be introduced at that time. Thinking about implementation details at this stage constrains the requirements elicitation process and as well as the requirements being elicited. Furthermore, these alternative options are truly solution suggestions that are part of design, not requirements

*Procedure:* Recognize when solution suggestions are made during requirements elicitation.

*Method:* Remove solution suggestions from the requirements document (artifacts) in case these have already been included. Determine the impact of solution suggestions on other requirements gathered. If other requirements are impacted, these should also be adjusted accordingly. It may be necessary to "re-discover/re-elicit" impacted requirements. The requirements engineer may not want to discard the solution suggestions removed from the requirements document, but file these in a special document for possible usage during design.

***Note:*** If this method has had to be used, the intent of the requirements elicitation process had been compromised. Care should be taken so that the same "mistakes" are avoided in the future.

**Scenario 2**: During a requirements elicitation meeting the discussion focus shifts from the stated subject/objective of the meeting

*Problem:* A successful elicitation meeting is one that meets its stated objectives. A significant portion of those objectives is defined in terms of the subject(s) to be addressed during the meeting. The intent, therefore, is to keep the meeting focused and moving towards a successful conclusion of activities. Nonetheless, because of the interrelatedness among system components and functionality, and because of a meeting participant's natural desire to understand how the system components being discussed impacts their operational domain, the meeting focus can easily shift to subject matter(s) not on the meeting agenda. For example, if the requirements for functional component A are the focus of the discussion, when its interface requirements to functional component B are addressed, focus of the discussion can easily shift to the operational aspect of component B. Not only does such a shift in focus impede achieving meeting objectives, but it also moves the discussion into a domain where the necessary background preparation may be inadequate.

*Procedure:* Recognize when the subject/objective of the meeting is no longer being address.

*Method:* Once it is determined that the subject matter being discussed is outside the set of objectives and subjects established prior to the meeting, the meeting moderator interrupts the discussion and informs the participants that the focus of the meeting has strayed from its stated objectives and subject matter. The moderator restates the original meeting objectives and subject emphasis, and then refocuses the discussion appropriately. Additionally, the moderator records the issues that prompted the shift and ensures that those issues will be addressed in a subsequent meeting.

**Scenario 3**: The customer is being vague in their characterization of system functionality and/or its particulars.

*Problem:* Vague descriptions leave too much latitude for inferences and misinterpretations. They often lead to ambiguous, unnecessary or incorrect requirements.

*Procedure:* Vague specifications incorporate characteristic phrases or terms. Meeting participants must maintain a constant vigilance for the use of terms or phrases like “possibly”, “maybe”, “might”, “can”, “I think so” or other similar phrases.

*Method:* Determine why the customer is using vague terms. Is it because the person describing the requirements (a) does not possess the requisite knowledge, or (b) is not sufficiently prepared to discuss the requirement at the required level of detail?

- In the event of (a) above the meeting moderator needs to determine who is the “right” person to consult. If that person is neither present nor readily accessible, the moderator makes a note to ensure that this person is included in the next meeting that address the requirement (or system characteristics) under consideration.
- In the event of (b) above, the meeting moderator reminds all participants that proper preparation is a prerequisite for a successful elicitation meeting and that if one cannot meet the preparation deadline then they mist notify the requirements engineer. In turn, the requirements engineer can either reschedule the meeting, or modify the intended objective and subject focus.

In either event, the meeting moderator should attempt to steer the elicitation meeting in a direction that does not rely on the “missing” information, yet in one that still conforms to stated meeting objectives and subject matter.

**Scenario 4**: While describing a particular aspect of the proposed system the speaker is often interrupted "mid-sentence."

*Problem:* When a person is in the process of describing a system characteristic or functionality, constant interruptions increase the probability of losing the current "train of thought." As a consequence, such interruptions can lead to incomplete or missing requirements. While we recognize that speaker interruptions are sometimes necessary, our objective must be to minimize the frequency and adverse impact that such interruptions can have.

*Procedure:* Recognizing dialogue interruptions is one of the more easier procedural tasks – such interruptions are self-evident. Their detection is an assigned task for all meeting participants, but in particular, for the session moderator.

*Method:* Both the frequency and adverse impact of "mid-sentence" speaker interruptions can be reduced through what we call "controlled interrupts." That is:

- Meeting participants are reminded to (a) record questions and issues using the "Silent Parking Lot" (see Appendix E), and then (b) request clarification or additional expansion at the appropriate time. In decreasing order of preference, the more appropriate times are (1) at the end of a topical matter, (2) at the end of a thought, or (3) at the end of a sentence.
- The interrupted speaker takes *intentional* steps to ensure that the discussion continues along the same line after the interruption is handled.

Although often viewed as having negative impacts, we note that interruptions can also have a positive impact on a discussion. For example, interruptions in the form of questions can contribute to improving the discussion by indicating to the speaker that specific issues or topics need clarification. We are of the opinion, therefore, that actively recognizing and controlling interruptions has more distinctive advantages than simply forbidden them.

## Appendix D. RGM Checklist for Exit Criteria

- ❑ The review for the latest requirements elicitation meeting has been completed
- ❑ All requirements have been found to be testable and/or verifiable, or were applicable noted specifically as non-testable/verifiable
- ❑ Each requirement is unambiguous
- ❑ Each requirement is accurate
- ❑ Each requirement is necessary
- ❑ The set of requirements is consistent
- ❑ The set of requirements is complete
- ❑ The current set of requirements has been evaluated with an automated requirements assessment tool if available (e.g., ARM [43]]
- ❑ No new issues have been identified during the most recent review
- ❑ No issues remain unresolved on the issues log
- ❑ The customer and requirements engineer have had a chance to paraphrase requirements to ensure customer understanding and that the customer's intent has been captured
- ❑ All requirements have been put under requirements management
- ❑ The requirements engineer agrees the current set of requirements are complete and ready for validation
- ❑ The customer agrees the current set of requirements meets all expectations and intentions and the project can move forward to requirements analysis or validation

# Appendix E. RGM Artifacts

## E1 MEETING NOTIFICATION

Meetings between the customer and the requirements engineer must be announced as part of the preparation sub-phase. A formal meeting notification shall be distributed to all identified participants to attend the next meeting, as well as other stakeholders that need to be informed. Several crucial components make up the formal meeting notification. The requirements engineer (or person making the announcement) must use the meeting notification template below. The information to be provided consists of:

- Date/Time: A preferred date format is Month Day, Year (e.g., February 31, 2000). The time should indicate A.M. and P.M. hours (or possible 24h format if international participants will be involved) as well as time zones should participants travel or dial in for video/telephone conferences. Starting and ending times for the meeting must be given and must be adhered to.
- Place: Give as much detail as possible to make it easy for participants to find the meeting place (this can help in making sure everyone arrives on time and is not searching for the room). If in multi-story building and not every participant is familiar with this building, give information such as elevators to use, which floor, which wing, etc.
- Contact info at meeting place: Participants should have a means to contact the meeting place directly. This is necessary so that someone who is delayed or lost can inform the other participants without them having to guess at their whereabouts. This is also important if an outside person needs to contact a meeting participant.
    - Meeting Subject: The meeting subject must be placed in the meeting notification.
    - Meeting Objective: The meeting objective must be placed in the meeting notification.
    - Meeting Detail Level: The detail level of the meeting must be included in the meeting notification.
    - Participants/Responsibilities: Identify who and in what responsibilities meeting attendees have that will participate. This enables all meeting participants to know

exactly who and how many will be participating (important if artifact copies are to be distributed). It is not necessary to include contact information (phone, email, address) with the meeting notification as this information is available from the project participant contact list.

- Issues: In the meeting notification all major issues to be discussed must be listed together with the identification of the individual(s) handling each issue.
- Contact Information: The person sending out the meeting notification should include contact information for the meeting participants to be able to respond to the notification.

The above items must be included with every meeting notification. The requirements engineer or person responsible for setting up a meeting may decide to include other pertinent information. The importance of everyone preparing on time for the meeting, and notifying the person setting up the meeting if this is not possible, is stressed during the setup phase with the customer. To remind all meeting participants a notice to this affect must be included with the meeting notification as shown.

Meeting Notification:

Date/Time:

Place:

Contact info at meeting place (such as a phone number):

Meeting Subject:

Meeting Objective:

Meeting Detail Level: (Overview, High Level, Low Level, Elaboration, Reification, etc.)

Participants and Responsibilities:

1.

2.

….

Issues (identify participants requested to handle each issue):

1.

2.

….

Please be advised that all issues should be prepared and if necessary distributed to all participants __ days prior to the meeting. If for some reason you will not be able to prepare in a timely fashion, cannot attend this meeting, or for any other question please notify ASAP.

Contact Name:

Contact Address:

Contact Phone:

Contact Email:

## E2 PARTICIPANT/ISSUE NOTIFICATION

During the evaluation sub-phase a requirements review of the newly identified requirements is conducted. Following this review the requirements engineer and customer will have formed a list of new issues to be resolved during the next iteration. Accordingly, personnel with appropriate background are identified and sent the Participant/Issue notification. This gives the identified participants a chance to suggest better "experts" for their assigned issues, or to help open communication channels between participants working on related issues. The template below identifies items that must be included in a participant/issues assignment notification. The notification itself can be distributed by email, interoffice mail, or regular mail, but not by voicemail alone.

Participant/Issues Assignment Notification:

Suggested Meeting Date/Time:
Suggested Meeting Place:

Assignments Due Date:

Participants and Roles:
1.
2.
….
Issues (identify participants requested to handle each issue):
1.
2.
….

Please be advised that all issues should be prepared and if necessary distributed to all participants __ days prior to the meeting. If for some reason you will not be able to prepare in a timely fashion, cannot attend this meeting, or for any other question please notify ASAP.

Contact Name:
Contact Address:
Contact Phone:
Contact Email:

## E3 SILENT PARKING LOT

**SILENT PARKNG LOT**

Project/Project Number:_____

Date:_____

Name and Contact information:_____

Please use this silent parking lot to write down your thoughts and ideas during the meeting before you can verbally express these (to minimize interruptions to the discussion flow). All silent parking lots will be collected after the meeting by the meeting facilitator or project manager and reviewed for remaining open issues. Please indicate those items that require follow-up communication with you.

Try to number items as you go.

## E4 UNSTRUCTURED REQUIREMENTS LIST (USRL) REVIEW

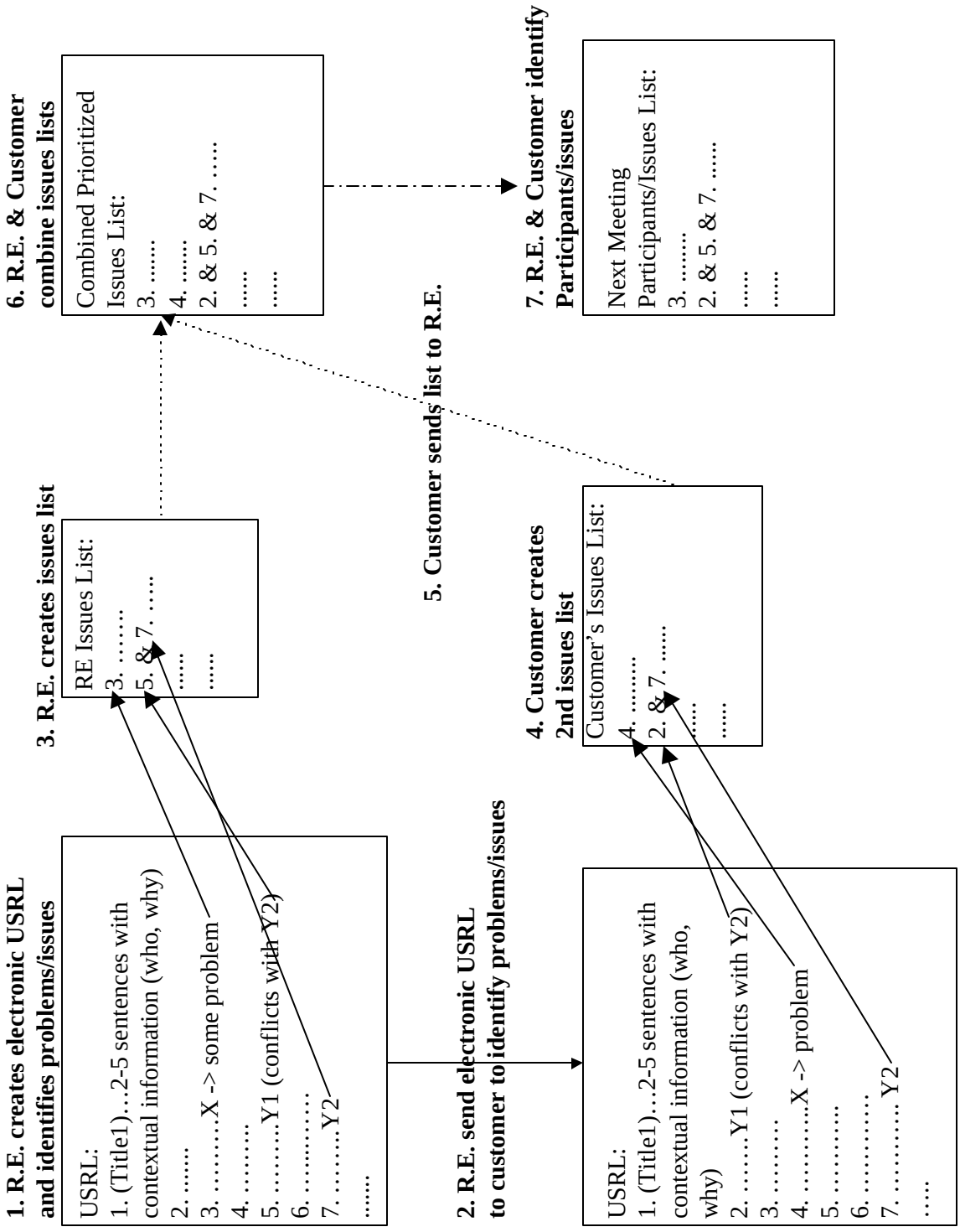
The evaluation sub-phase within requirements capturing immediately follows each elicitation meeting. The requirements engineer should schedule time following the elicitation meeting to review the identified requirements. The requirements engineer should allow for at least one hour to conduct this review. The requirements review should be done alone or possibly with the meeting moderator, but never with the customer. The requirements review consists of the following steps and should include transposing any handwritten requirements to electronic form (i.e. a Requirements Management Tool):

1. For each requirement elicited write it in its "purest" form (identification of requirement, requirements category, etc.).
2. Include additional 2-3 sentences or more if necessary, with each requirement. This additional information should be precise and give enough detail on the requirement such that it can be completely understood at the time that the SRS is written (which can be several weeks or months from the time this requirement was elicited). If relevant, information that can be included are the names and/or positions of those stakeholders requesting or that are impacted by this requirement as well as the reasons for the requirement.
3. Conflicting or Questionable Requirements: As the requirements review is conducted the requirements engineer will encounter requirements elicited in the interaction meeting, which require further exploration. These requirements fall into one of two categories: a) normal requirements that require either progressive elaboration or reification, and b) problem requirements that require clarification and resolution. Some of the "problem" requirements are based on one or more conflicting requirements that have to be resolved. Other "problem" requirements are those that the requirements engineer does not completely understand and require clarification. All requirements needing further resolution must be transferred to an "Issues List."
4. The requirements engineer must give the edited unstructured requirements list (USRL) to the customer for review.
5. The customer is asked to review the edited USRL list for correctness and, like the requirements engineer, is expected to generate an "Issues List." **Note:** The "Is-

sues List" the requirements engineer generates must not be given to the customer, as it will influence their view of the unstructured requirements list.

6. The customer must prioritize the "Issues List."
7. The customer returns the "Issues List" to the requirements engineer upon completion.
8. The requirements engineer combines both Issues Lists, and prioritizes the combined issues list (the final prioritization may be done with input from the customer).
9. The requirements engineer (and possibly the customer) decides on the top priority issues to be discussed during the next interaction session. These issues must be communicated in the next meeting notification, and participants must be identified who can help resolve the issues.

## E5 EVALUATION SUB-PHASE REVIEW WORKFLOW



# Appendix F. Empirical Study Data

## F1 RGM GROUP PROJECT DATA

**Project 1 (RGM Group, large prj.)**

| | Original Date | # PD | Actual Date | # AD | # SD | |
|---|---|---|---|---|---|---|
| Project Start | 11/20/2000 | | 11/20/2000 | | | |
| Definition | 12/20/2000 | 31 | 12/20/2000 | 31 | 0 | |
| Design | 1/5/2001 | 16 | 1/12/2001 | 23 | 7 | |
| Implementation | 2/28/2001 | 54 | 2/28/2001 | 47 | -7 | % Slippage |
| Total Days | | 101 | | 101 | 0 | 0% |

**Project 2 (RGM Group, large prj.)**

| | Original Date | # PD | Actual Date | # AD | # SD | |
|---|---|---|---|---|---|---|
| Project Start | 10/16/2000 | | 10/16/2000 | | | |
| Definition | 11/15/2000 | 31 | 11/15/2000 | 31 | 0 | |
| Design | 12/31/2000 | 46 | 12/22/2000 | 37 | -9 | |
| Implementation | 1/2/2001 | 2 | 1/2/2001 | 11 | 9 | % Slippage |
| Total Days | | 79 | | 79 | 0 | 0% |

**Project 4 (RGM Group, small prj.)**

| | Original Date | # PD | Actual Date | # AD | # SD | |
|---|---|---|---|---|---|---|
| Project Start | 10/24/2000 | | 10/24/2000 | | | |
| Definition | 11/17/2000 | 25 | 11/17/2000 | 25 | 0 | |
| Design | 1/15/2001 | 59 | 1/12/2001 | 56 | -3 | |
| Implementation | 1/20/2001 | 5 | 2/1/2001 | 20 | 15 | % Slippage |
| Total Days | | 89 | | 101 | 12 | 13% |

**Project 5 (RGM Group, small prj.)**

| | Original Date | # PD | Actual Date | # AD | # SD | |
|---|---|---|---|---|---|---|
| Project Start | 10/23/2000 | | 10/23/2000 | | | |
| Definition | 11/17/2000 | 26 | 11/17/2000 | 26 | 0 | |
| Design | 1/8/2001 | 52 | 1/7/2001 | 51 | -1 | |
| Implementation | 1/22/2001 | 14 | 2/15/2001 | 39 | 25 | % Slippage |
| Total Days | | 92 | | 116 | 24 | 26% |

**Legend:**

*Original Date* - (OD) Date the phase was supposed to be finished.

*Actual Date* - (AD) Date the phase did finish.

*# PD* - #(P)hase (D)ays - Number of original calendar days this phase should have taken to finish.

*#AD* - #(A)ctual (D)ays - Number of actual calendar days this phase took to finish.

*#SD* - # (S)lippage (D)ays - The number of days a phase went over/under the original estimated phase finish day.

**Formulae:**

$\#SD_{Pi} = \#AD_{PI} - \#PD_{PI}$

Total # Projected Calendar Days for Project (Total #PD) = $\Sigma\ \#PD_{PI}$

Total # Actual Calendar Days for Project (Total #AD) = $\Sigma\ \#AD_{PI}$

Total # Slip Days (Total #SD) = Total #AD - Total #PD

%Slippage = Total #SD / Total #PD

## F2 NON-RGM GROUP PROJECT DATA

**Project 3 (Non-RGM Group, large prj.)**

| | Original Date | # PD | Actual Date | # AD | # SD | |
|---|---|---|---|---|---|---|
| Project Start Day | 6/5/2000 | | 6/5/2000 | | | |
| Definition | 7/1/2000 | 27 | 7/28/2000 | 54 | 27 | |
| Design | 7/1/2000 | 0 | 2/28/2001 | 215 | 215 | |
| Implementation | 9/1/2000 | 44 | 4/5/2001 | 36 | -8 | % Slippage |
| Total Days | | 71 | | 305 | 234 | 330% |

**Project 6 (Non-RGM Group, small prj.)**

| | Original Date | # PD | Actual Date | # AD | # SD | |
|---|---|---|---|---|---|---|
| Project Start Day | 8/14/2000 | | 8/14/2000 | | | |
| Definition | 9/1/2000 | 19 | 9/20/2000 | 28 | 9 | |
| Design | 10/1/2000 | 30 | 10/18/2000 | 28 | -2 | |
| Implementation | 3/1/2001 | 151 | 2/15/2001 | 120 | -31 | % Slippage |
| Total Days | | 200 | | 176 | -24 | -12% |

**Project 7 (Non-RGM Group, small prj.)**

| | Original Date | # PD | Actual Date | # AD | # SD | |
|---|---|---|---|---|---|---|
| Project Start Day | 7/31/2000 | | 7/31/2000 | | | |
| Definition | 8/14/2000 | 15 | 10/5/2000 | 67 | 52 | |
| Design | 8/28/2000 | 14 | 10/20/2000 | 15 | 1 | |
| Implementation | 9/15/2000 | 18 | 11/12/2000 | 23 | 5 | % Slippage |
| Total Days | | 47 | | 105 | 58 | 123% |

**Legend:**

*Original Date* - (OD) Date the phase was supposed to be finished.

*Actual Date* - (AD) Date the phase did finish.

*# PD* - #(P)hase (D)ays - Number of original calendar days this phase should have taken to finish.

*#AD* - #(A)ctual (D)ays - Number of actual calendar days this phase took to finish.

*#SD* - # (S)lippage (D)ays - The number of days a phase went over/under the original estimated phase finish day.

**Formulae:**

$\#SD_{Pi} = \#AD_{PI} - \#PD_{PI}$

Total # Projected Workdays for Project (Total #PD) = $\Sigma\ \#PD_{PI}$

Total # Actual Workdays for Project (Total #AD) = $\Sigma\ \#AD_{PI}$

Total # Slip Days (Total #SD) = Total #AD - Total #PD

%Slippage = Total #SD / Total #PD

## Vita – Markus K. Gröner

Markus K. Gröner was born July 20, 1967 in Hanau/Main, Germany. He received his B.S. in Computer and Information Sciences from the University of South Alabama (USA) in June 1990. Software Engineering became his focus of interest during his Masters studies, reflected in his Master's Thesis "Joint Participation in Software Design". He received his M.S. in Computer Science from the University of South Alabama in summer 1994. During his graduate time at USA he held a teaching assistantship and was also president of the Association for Computing Machinery (ACM) Student Chapter from 1993 - 1994.

He began his Ph.D. studies at Virginia Polytechnic Institute and State University (VT) in August 1994. His area of interest is in Requirements Engineering with focus on Requirements Specification improvement. He held a graduate research assistant from August 1994 - August 1996, conducting research on a grant by NASA focusing on the Software Engineering Evaluation System. In August 1996 - May 1996 he received a graduate teaching assistantship instructing freshmen Operating Systems Tools classes. In the summer of 1997 and 1998 he taught Computer Literacy, Internet Introduction, and Operating Systems Tools classes at VT. In fall 1998 he was an instructor at Radford University teaching Software Engineering and Operating Systems courses. He also held a research assistant position in summer 1998 through August 1999 with the PCs for Families research project at VT funded by the DOE. Markus is a member of both ACM and IEEE.

Markus lives with his wife Miriam and their five cats in Richmond, VA. He can be reached at markus@groeners.org.

_____

Markus K. Gröner