An Operational Model Supporting
The Generation of Requirements That Capture Customer Intent

Markus K. Gröner James D. Arthur
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061-0106

{groener, arthur}@vt.edu

Abstract: Product quality is a direct reflection of how well it meets the customer's needs. Hence, the ability to capture customer requirements correctly and succinctly is paramount. Unfortunately, within software development frameworks the gathering of requirements is one of the more ill-defined and least structured processes. To address this inadequacy, our paper (a) proposes a refinement (or decomposition) to the requirements generation phase that admits to detailed examination and promotes a better understanding of where requirements generation goes wrong, (b) using the results of a preliminary study, identifies where and how miscommunication and requirements omission occur, and (c) outlines a more structured process that can be applied early in the requirements generation process to minimize requirement faults.

Keywords: Requirements Identification, Requirements Generation, Customer Intent, Requirements Engineering, Software Engineering, Software Methodology

Author biographies

Markus K. Gröner is currently pursuing his Ph.D. at Virginia Tech (VPI&SU). He received his B.S. and M.S. in Computer and Information Sciences from the University of South Alabama (USA) in 1991 and 1994. His research interest is rooted in Software Engineering with a special emphasis on (a) establishing an error-resistant communication framework between Clients and Software Engineers, and (b) improving Software Requirement Specifications through better IV&V methods. Mr. Gröner has held teaching and research assistantships throughout his graduate career. He is a member of both the ACM and IEEE societies. Mr. Gröner can be contacted at Virginia Tech, Department of Computer Science, 660 McBryde Hall, Blacksburg, VA, 24061. His email address is groener@vt.edu; his telephone number is (540) 231-5853.

James D. Arthur received the B.S. and M.A. in mathematics from the University of North Carolina, the M.S. from the University of Houston, and the M.S. and Ph.D. from Purdue University. He has been involved in software engineering research for the past fifteen years. His research interests encompass requirements engineering, IV&V, software quality assessment, user environments, and parallel computation. He is currently an Associate Professor of Computer Science at Virginia Tech in Blacksburg, VA. Dr. Arthur can be contacted at Virginia Tech, The Department of Computer Science, 660 McBryde Hall, Blacksburg VA, 24061. His e-mail address is arthur@vt.edu; his telephone number is (540) 231-7538.

An Operational Model Supporting
The Generation of Requirements that Capture Customer Intent

1. Introduction

A successful software development effort, i.e. one resulting in a quality product delivered on schedule and within budget, depends on a well-defined software development process. Such a process is guided by procedures and methods outlined in models like the Waterfall Model [1] and the Spiral Model [2], as well as development approaches such as rapid or throw-away prototyping [3] [4]. The objective in using these procedures and methods is to promote product correctness and to provide qualitative guidance during development. Our research and experience have shown, however, that no matter how rigorous the development process is, the delivered product is only as good as the set of requirements by which it is defined. More specifically, a delivered software product may possess all of the quality aspects induced through an effective software engineering process, yet not fulfill the most important objective -- perform as intended by the customer.

The primary task of the requirements engineer is to capture accurately the requirements expressed by the customer. To do so, the real challenge emerges -- recognizing inconsistent, wrong, and/or implied requirements. Contributing to that challenge is the observation that often the customer (a) has only a minimal understanding of the many functions that the proposed system is to support, (b) lacks experience in identifying and expressing desirables as requirements, and (c) states desirables in terms that lead to misleading or incorrect inferences by the requirements engineer [5]. In similar ways, the requirements engineer also contributes to the challenge. Consequently, we often fail in or find it very difficult to achieve our task of capturing requirements that *truly* reflect the customer's intent and expectations.

Models and methodologies that assist *both* the requirements engineer and the customer in the generation process are fundamental to capturing requirements accurately. Unfortunately, finding such a model or methodology that is *effective* is difficult indeed. Existing models like Participatory Design (PD) [6] [7] [8] and Joint Application Design (JAD) [9, 10] expand on conventional software development lifecycle models. In particular, they focus on reducing the disparity in domain knowledge between the requirements engineer and customer, respectively. Both PD and JAD prescribe a closer working relationship between the requirements engineer and customer – which, expectantly, should result in the generation of a better set of software development artifacts. The major obstacle to the acceptance of PD and JAD by industry, however, has been its high cost in terms of time commitment for both the customer and development personnel [7]. Nonetheless, the contributions of these (and other) approaches, *and* their apparent pitfalls, underscore the need for a cost-effective approach to improving the requirements generation process, and ultimately, the quality of the product.

In response to the above observations, this paper focuses on the following question: *"How does one evolve requirements that capture the intent of the customer?"* The results of an initial study and on-going research indicate that one answer lies in providing structure and guidance during the requirements definition phase. Reflecting those results, we propose (a) a framework that provides structure to the requirements definition process, and (b) an integrated methodology that guides the generation process. The framework partitions the conventional requirements definition phase into three well-defined mini-phases:

- A *setup* mini-phase that governs all activities up to and including the initial meeting between the customer and the requirements engineer,
- A *requirements capturing* mini-phase that focuses on the identification, recording and refinement of requirements, and
- A *transformation* mini-phase that guides the process by which the requirements and context information produced in the preceding mini-phase are transformed into a software requirements specification document (SRS).

The methodology consists of protocols, procedures and methods that assist in requirements elicitation, recording and analysis. In particular the methodology focuses on two aspects of requirements generation:

- Providing “tried and true” guidance that has a high probability of success, and
- Incorporating procedures for identifying characteristics indicative of problems and suggesting correction methods.

In effect, the framework and methodology have been purposefully designed to (a) add structure and control to the requirements definition phase, (b) support the accurate capturing of requirements, and we conjecture, (c) reduce the cost of an effective requirements generation process.

The structure of the remainder of the paper is as follows. Section 2 provides an introduction to the framework overlaying the conventional requirements definition phase. The figure provided and the corresponding discussion illustrate *where* guidelines and protocols fit within the framework and describe *how* they support improved requirements generation. Section 3 presents a detailed discussion of the second mini-phase: *requirements capturing*. Section 4 describes the three primary components of our proposed methodology and discusses how they span the phases of requirements definition. Results from a prior study are used to illustrate the potential utility of the methodology. Lastly, section 5 provides a summarization of our proposed model and a discussion of current work in progress.

2. The Operational Model: Framework, Guidelines and Protocols

Given the conventional Waterfall Model (or one of its many variants), five development phases are most prominent: requirements, design, implementation, testing, and maintenance. Moreover, we often find one or more of these phases further decomposed to emphasize important sub-components. For example, it is common to find design split into a high- and a low-level design phase, as well as implementation divided into coding and unit-testing. The requirements generation phase is composed of two primary activities - requirements definition and requirements analysis. Because of inconsistent terminology [11], however, requirements generation is often equated with only requirements analysis. This view, unfortunately, promotes the misconception that requirements definition is the less significant of the two. Clearly, such implications run counter to the observations that "a system is only as good as the requirements from which it is built" and "requirements are only as good as the extent to which they reflect the customer's needs and intent."

To address the misconception noted above and to promote the generation of requirements that better reflect the customer's intent, we propose a refinement of the requirements definition phase that imposes structure and methodology. The three major components of the refinement are (a) a structured framework that partitions the requirements definition activity into three mini-phases, (b) a set of guidelines and protocols to suggest and structure activities within the framework, and (c) procedures and methods that assist in requirements definition.

To start, this section provides an overview of the proposed framework. Following the overview, we outline the roles of guidelines and protocols as related to structuring the framework.

2.1 The Proposed Framework

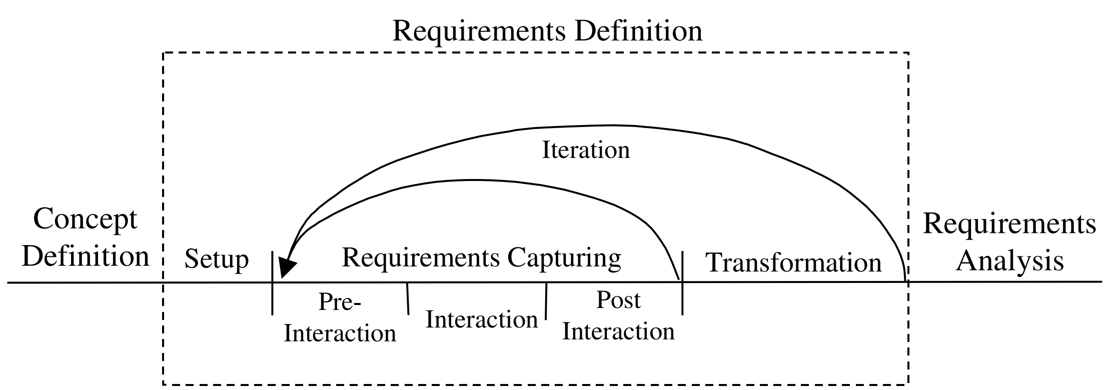

Figure 1: Proposed Framework

As illustrated in Figure 1, the proposed framework for requirements definition is comprised of three mini-phases: the setup, requirements capturing, and transformation. Furthermore, requirements capturing is partitioned into three additional components: pre-interaction,

interaction and post-interaction. The structure of the framework underscores an important aspect of requirements definition: it consists of several distinct activities with each having its own independent set of objectives. The framework structure also illustrates that the requirements definition process is not necessarily linear in activities. That is, requirements definition is (or should be) an iterative process that constantly refines and elaborates on existing requirements. An outline of the three mini-phases is provided below.

The Three Mini-Phases

(1) The "setup" phase is the first of the mini-phases. It governs all activities leading up to and including the initial meeting between the customer and the requirements engineer. In particular, the setup mini-phase outlines each participant's initial responsibilities prior to beginning requirements capturing. For example, the customer is to provide a high-level overview of the existing and proposed system; the requirements engineer is to instruct the customer about the process underlying requirements generation.

(2) The second mini-phase, "requirements capturing", focuses on those activities where requirements are being identified, recorded and refined. Within this mini-phase the customer and requirements engineer prepare documents for meetings, attend well-structured and controlled meetings, examine artifacts from those meetings, and decide if additional meetings are necessary. The requirements capturing mini-phase is iterative in nature, allowing the refinement of requirements through progressive elaboration and reification [12]. Artifacts produced during this mini-phase include an unstructured software requirements list (USRL) and supporting contextual information.

(3) The "transformation" mini-phase is third in our partitioning and guides the process by which the USRL and contextual information (defined during the requirements capturing mini-phase) are transformed into a software requirements specification document (SRS). Initially requirements are grouped together based on similarity characteristics; connections between those groups are then established. Among other criteria, requirements are examined for completeness, accuracy, understandability, and their relationship to other requirements. If major problems are identified during this phase, an additional iteration through the requirements capturing mini-phase is initiated.

In an effort to minimize duplication of effort between the transformation mini-phase and requirements analysis, the analysis phase should exclude or significantly limit the effort allocated to overlapping activities.

The framework presented above outlines an organization of activities necessary for effective requirements generation. Those organization and attendant activities are not, however, sufficient. In particular, clear and unambiguous guidance reflecting the objectives and constraints within and among the individual activities must be provided. The guidelines and protocols described below are integral elements of our operational model, and provide that needed guidance.

2.2 Guidelines

Guidelines are used to familiarize the participants with the requirements definition process, and to *suggest* activities that help ensure a successful requirements definition process. To help educate the participants, guidelines include overviews of

- the requirements definition process, its components, and activities,
- what results can be expected when going through the process,
- how the participants are involved in the process, and
- how each participant should prepare for the definition process.

More generally, the guidelines provide suggestions for supportive activities that (a) span all mini-phases, or (b) focus on individual ones.

Cautionary Note: Guidelines must be written in a language that can be understood by all participants. The use of technical terminology should be limited and must be explained in a glossary. In effect, guidelines are a reference for all participants throughout the entire requirements definition process and must be written accordingly.

2.3 Protocols

Protocols define the structure imposed on the framework. Unlike guidelines that offer guidance (adherence is optional) to the participant in the process, protocols establish constraints that define the structure (adherence is required). More specifically, given the objectives of any particular mini-phase, protocols help to ensure that they are met by keeping all participants operating within well-defined "structural boundaries." Similar to guidelines, certain protocols permeate the framework while others are specific to individual mini-phases. For example, protocols specific to the requirements capturing mini-phase state that before every interaction meeting (a) the subject and objective of that meeting must be established, and (b) all participants and their roles must be identified.

Clearly, protocols state "what must be done." We acknowledge, however, that they are only words on paper. If they are not followed, one *does not realize* their beneficial contributions.

3. Phases

In section 2 we introduce the framework and outline its three mini-phases. We now present a more detailed description of the framework's first three mini-phases: (1) setup, (2) requirements capturing, and (3) requirements transformation. The following discussion focuses on problems identified through experience and research that can occur during these mini-phases. We introduce procedures and methods designed to prevent or eliminate the problems associated with the requirements capturing process. Although the problems discussed in this paper are not specific to any one particular domain, they do represent some of the more critical ones. We contend that even after an application domain is established, little (if any) tailoring of our proposed procedures and methods will be required.

3.1 Setup

As shown in Figure 1, the setup mini-phase follows concept definition and focuses on defining the working relationship between customer and requirements engineer. The objectives of the setup mini-phase are (a) to introduce the customer to the requirements definition process, (b) to provide the requirements engineer with an overview of the customer's problem domain and needs, and (c) to help prepare the customer for involvement in the requirements definition process. The following five items outline a set of activities and documents designed to achieve these objectives.

1. *Guidelines*: The customer is given a set of guidelines that introduce the requirements generation process and define the working relationship between the customer and the requirements engineer.
2. *Domain overview*: The customer provides an introduction of his/her work environment and system application domain to the requirements engineer. This is achieved through presentations, guided tours and/or meetings with company personnel. The requirements engineer, who should have some initial knowledge of the domain, acquires additional familiarity with the customer's problem domain.
3. *Process overview*: The requirements engineer provides the customer with (a) an overview of the software development, (b) a description of the requirements definition process, and (c) instructions on how to prepare material for the upcoming requirements capturing process.
4. *System overview*: The customer provides the requirements engineer with documentation to (a) the current system, and (b) the proposed system, including possible feasibility studies, et cetera. Furthermore the customer presents an overview of what he/she believes to be the more critical or difficult parts of the system.
5. *Q/A session*: The Q/A session provides the customer and requirements engineer with an opportunity to follow up on elements of the setup mini-phase that are still unclear. The customer may have further questions pertaining to parts of the process. The

requirements engineer may ask the customer for clarification on items discussed or presented.

At the conclusion of the setup mini-phase the customer is expected to have gained an appreciation of the requirements generation process and knows how to prepare (material and questions) for the subsequent mini-phases. The requirements engineer is expected to have gained familiarity with the application domain of the customer and the needs for the new system. Following the setup mini-phase the customer and requirements engineer should be able to answer "yes" to the following questions:

1) Have the rudiments of the problem been communicated?
2) Are the fundamental issues understood?
3) Do we understand how to prepare for the upcoming interaction session?

We emphasize that preparing the customer well for the process is the responsibility of the requirements engineer.

3.2 Requirements Capturing

The requirements capturing mini-phase, as shown in Figure 1, follows setup. The objectives of the requirements capturing mini-phase are (a) to provide a structure within which requirements are accurately identified, and (b) to support the refinement of identified requirements through iteration. How these two objectives are achieved is documented in the following three sub-sections. The discussions reflect a further decomposition of the requirements capturing mini-phase into three working sub-phases: (1) pre-interaction, (2) interaction, and (3) post-interaction.

In reaching the overall objective of requirements capturing we describe objectives and tasks associated with each sub-phase. We also explain how refinement of requirements is realized through iteration over the requirements capturing mini-phase.

3.2.1 Pre-Interaction

The pre-interaction sub-phase is initially entered from the setup mini-phase, and subsequently by iteration from the post-interaction mini-phase. The objective of the pre-interaction sub-phase is (a) to examine issues stemming from either the setup or post-interaction mini-phase, (b) to resolve those issues, and (c) to prepare for the interaction meeting. Part (c) is achieved by documenting solutions to resolved issues and by recording problems and questions on issues that are not resolved.

Entering from setup:

Entering pre-interaction from setup occurs only once during the requirements capturing process. Initial tasks to complete include:

- establishing the subject matter (system components) and objective (level of detail) of the upcoming interaction meeting,
- identifying the major participants and their roles,
- conveying to all participants the issues to be addressed during the interaction meeting, and
- ensuring that all participants clearly understand how to prepare for that meeting.

Entering from post-interaction:

Entering pre-interaction from post-interaction occurs with each iteration of the requirements capturing mini-phase. The initial tasks to address are:

- establishing the subject matter for the upcoming interaction meeting,
- identifying the reason for iteration, including one or more of:
    - unresolved issues from previously identified requirements
    - incomplete or missing high level requirements, requiring further requirements identification (breadth)
    - lack of detail in previously identified requirements (depth)
- establishing the expected detail at which the material is to be presented, e.g., overview, mid-level, or detailed.
- identifying additional participants (possibly domain experts) and their roles,
- conveying to all participants those issues to be addressed during the interaction meeting, and
- ensuring that all participants clearly understand how to prepare for the meeting.

After completing the initial set of tasks, the participants (usually the customer and requirements engineer) of the upcoming meeting prepare the appropriate material. During this preparation period, participants are required to:

- separately address unresolved issues and document their solutions,
- record unresolved issues that require further investigation during the interaction meeting, and

- indicate their readiness for the interaction sub-phase - Michael Fagan [13] suggests that it is better to postpone an inspection session than to enter it unprepared.

In general, communication between customer and requirements engineer is more focused on: (a) asking simple questions of understanding, and (b) determining meeting times and places.

The overarching goal of the pre-interaction mini-phase is to ensure that each participant carefully prepares for the ensuing discussions so that requirements can be succinctly conveyed during the interaction mini-phase.

3.2.2 Interaction

The tasks and problems we have discussed in the previous sections deal primarily with preparation, and making sure that all participants understand what is expected of them. Being well prepared contributes to more successful interaction sessions, but it does not prevent participants from making mistakes. Our experience and research has shown that most errors in communicating and capturing requirements are committed during interaction sessions.

The main objective of the interaction sessions is to enable the requirements engineer to accurately identify and record system requirements as expressed by the customer. In identifying requirements, the problems and associated tasks facing the requirements engineer differ from those of the customer because of the difference in their respective roles: the customer's role is to *convey* information relevant to system development, and the requirement engineer's role is to *capture* information and context accurately. Differing roles and differing session goals require specialized tasks to help answer questions and solve problems. We cannot therefore provide simple guidance for the interaction sub-phase, but instead, must rely on a combination of guidelines, protocols, and a methodology to detect and prevent problems. In the remainder of this "interaction session" discussion we introduce some of the inherent problems addressed by protocols and our proposed methodology.

As outlined in section 2, structure imposition and enforcement is the focus of guidelines, and more specifically, protocols. In this section we discuss four protocols that help provide a defining basis for interaction meetings.

- *Subject of Meeting*: Protocol requires that the subject matter for each interaction meeting be identified before that meeting can occur, and thereby, provides a discussion focus for the customer and requirements engineer.
- *Objective of Meeting*: As discussed in the pre-interaction sub-phase, the meeting objective for every interaction meeting must be established *a priori*. This, too, is mandated by the protocol. During the interaction session the customer and the requirements engineer must work toward meeting that objective.
- *Iteration Reason*: We have previously stated that after the initial entry into the interaction phase, all subsequent meetings stem from the need to "iterate" on some aspect of the problem. Protocol prescribes that one or more of the following motivations guide each subsequent interaction session:
    (a) unresolved issues – such issues are reflective of requirements being stated incorrectly, incomplete, or incomprehensibly. These unresolved issues are documented and conveyed to all participants. Some, or all, of the unresolved

issues and their solutions are topics for discussion during the next interaction session.

(b) progressive elaboration – not all requirements have been identified in previous interactions. The set of requirements is still incomplete and does not describe the complete system. Further requirements need to be identified.

(c) refinement – requirements have been identified but further detail on these requirements needs to be established.

- *Length of Meeting*: Protocol stipulates that meeting lengths must be kept to comfortable length. The actual length of a meeting should not be more than one to two hours, often depending on the "difficulty level" of the material being discussed. Meetings must be effective and meet their objectives. Fagan [13] [14]and others [5] [15] [16] have discussed that effectiveness can only be achieved if those involved have a fresh mind and are not "tired of" the discussion. Having several shorter meetings of no more than one to two hours each helps one maintain a focus on the subject of the meeting, as time for "unnecessary" discussion is limited.

The items outlined above relate to the model's structure and are controlled by protocol. By design, they should complement rather than oppose each other. Problem categories that are not part of the structure but are more related to activities are addressed through methodological procedures and methods. We outline some of those categories below.

- *Problem Recognition*: During an interaction session, the requirements engineer, and to a lesser extent the customer, must: (a) recognize when problems occur, (b) be able to categorize the problem, and (c) based on that categorization, identify a solution to the problem. Using the procedures and methods defined in our proposed methodology, the customer and requirements engineer are able to do just that. During the interaction meeting it is important that when problems are encountered, they can be adequately addressed and resolved before proceeding further with requirements capturing. Identified problems not actively addressed *do not* resolve themselves.

- *Constraint Recognition*: All systems have constraints bounding the requirements. Most often these are time, money, and environment. Additional constraints are often imposed by the system, the customer, or the requirements engineer. For example, the requirements engineer might decide to use an Off-the-Shelf (OTS) component before the requirements capturing process is complete. For the sake of argument, suppose that the OTS component requires system inputs based on a pre-defined and "unnatural" format. Unfortunately, the requirements engineer is then forced to write requirements that fit the dictates of the OTS component. Hence, the customer and requirements engineer must recognize when unnecessary and inappropriate constraints are being imposed, and then, work to remove them.
- *Controlled Interrupts*: Our research has shown that dialog interruptions (and disruptions) are some of the more commonly occurring problems, and also the easiest to detect. In our study, the customers and requirements engineers did not seem to be aware of the interruptions. Yet, when analyzing interaction sessions we found a detrimental impact on requirements caused by those interruptions. The adverse impact stems from (a) an unintentional change in discussion topic, (b) early termination of a detailed discussion, and (c) the speaker forgetting what was going to be said. It is prudent that the customer and requirements engineer have methods at their disposal to

assist in recognizing detrimental interrupts and for handling them in a controlled fashion. We call this "controlled stop-and-go", in which the interrupted person takes active notice of the interruption, and in doing so, takes *intentional* steps to ensure that discussion continues along the same line after the interruption is handled.

We note, however, that not all interruptions have negative impact. For example, interruptions in the form of questions can contribute to improving the discussion by indicating to the speaker that topics need to be better articulated. We therefore believe that *actively* noticing and controlling interruptions has more advantages than disallowing interruptions, which seem to be "natural" to human communications.

- *Explicit/Implicit Requirements*: Explicit requirements are those requirements *stated clearly* during the requirements capturing phase. On the other hand, implicit requirements are those *implied* through parts of the conversation. If correctly capturing explicit requirements is difficult, determining implicit requirements must be a game of luck. A structured communication process that encourages the iterative refinement of requirements will certainly set the stage for identifying explicit *as well as implicit* requirements.

To summarize, from an interaction perspective we must not place constraints on *what* is discussed between participants. Instead, we need to constrain (or structure) the framework within which communications takes place. In doing so we allow communications to occur "naturally" within a well-defined boundary, while meeting the interaction session objectives.

We then employ specialized procedures to identify problems when they do occur, and through our methods resolve them.

3.2.3 Post-Interaction

The post-interaction sub-phase follows the interaction sub-phase (see Figure 1) and is the last sub-phase in requirements capturing. The objectives of the post-interaction sub-phase are: (a) to determine if the objectives of the immediately preceding interaction session have been met, and (b) to check if all exit criteria have been met to enter the transformation mini-phase (section 3.3).

To determine if the objectives of the interaction session have been met, the customer and requirements engineer review the artifacts (USRL and context information) from previous interaction sessions. More specifically, the artifacts are examined to ensure that:

(1) the subject matter identified and the focus of the preceding interaction sub-phase have been adequately addressed, and

(2) the results of that interaction session are documented at the appropriate level of detail.

During that examination, deviations from these objectives are documented as *unresolved issues*. The severity of those deviations is assessed and a decision is made either to re-iterate on those issues or to move towards the transformation mini-phase.

If the artifacts meet the above two objectives, then they have met the first of several exit criteria for the post-interaction sub-phase. The remaining exit criteria include:

- checking the USRL and related context information for completeness,

- determining if the requirements are well-stated,
- determining if the requirements are understood clearly by the customer – this includes an interaction session during which the customer reads the requirements and gives his/her verbal interpretation of them to the requirements engineer, and finally
- *ensuring that the requirements, without exception, meet the customer's intent*.

When all exit criteria are sufficiently met the transformation mini-phase is entered.

3.3 Transformation

As shown in Figure 1, the transformation mini-phase follows successful completion of the requirements capturing mini-phase. Furthermore, it is the last mini-phase that is part of the refined partitioning of the requirements definition phase within our framework, and precedes the conventional requirements analysis phase. The objective of the transformation mini-phase is to transform the USRL and related contextual information into an SRS. This objective is met by a two-step approach: (1) structuring requirements of USRL into an SRS based on the contextual information, and (2) determining if the SRS is complete. In the following section we describe the details of each of these two steps.

The task of the first step is to take the requirements of the USRL and the contextual information captured during requirements identification and create an SRS. This requires that the requirements engineer identifies functional-related inter-relationships among requirements and group them according to that functional cohesiveness.

In the second step of this approach the requirements engineer must check for completeness of the newly created "structured requirements specification." The following questions are used to guide completeness assessment:

1. For each functional group of requirements, has the source of its inputs and recipient of its output been identified?
2. Does the SRS give the complete "picture" of the system and are all parts of the SRS well understood?

During any part of the transformation process, if the requirements engineer discovers any problems, then based on the severity of those problems, the requirements engineer can

(a) request an extra iteration through the requirements capturing mini-phase (severe error), or

(b) resolves the problems in a less formal manner, e.g., a short meeting with the customer (minor error).

4. Methodology

The previous sections of this paper present the proposed framework, and describe the mini-phases and activities within this framework. As part of this discussion, we have introduced, without elaboration, the notion of procedures and methods. The rationale behind the procedures and methods is to direct the requirements generation process and assist in problem identification and resolution. These procedures and methods, as well as protocols, comprise our proposed methodology. In this section we present the objectives of the methodology and show how we achieve those objectives through the use of protocols, procedures and methods.

The objectives of the methodology are to enforce the structure needed for the requirements generation process, and to guide the customer and requirements engineer in problem identification and resolution. To achieve these objectives we have designed a two-prong approach:

(1) define the structure (and constraints) through protocols, within which the customer and requirements engineer must interact, and

(2) identify and resolve problems using procedures and methods, respectively.

Analogous to medicinal practice, our methodology provides one set of "preventive" measures and another set of "curative" ones.

4.1 Structure through Protocols

For the most part interaction between the customer and requirements engineer transpires without problems and with positive results. There are instances, however, where such interactions lead to situations that have the potential to adversely impact requirements generation. To minimize the possibility of problems occurring, our methodology includes protocols designed to "constrain" the interaction latitude provided to the customer and requirements engineer. In actuality, it is the protocols that define the framework's structure and boundaries to which one must adhere. Remaining within the boundaries of that structure helps focus the customer and requirements engineer on the tasks they must perform, which in turn, helps achieve the objectives of the requirements generation process.

Examples of structure imposed by protocols are:

(a) participant role identification – Protocol mandates that before any meeting occurs, all participants and their respective roles be identified. Role identification includes:
  - establishing the responsibilities and tasks of each participant during the interaction session (e.g., leader, recorder, presenter, etc.),
  - ensuring that a single person is designated as “in charge”,
  - assigning multiple roles to one participant during one interaction session (e.g. the leader is also the presenter), and
  - switching roles between participants in different interaction sessions.

(b) identification of subject matter and objective of meeting – Protocol also dictates that the subject matter and meeting objective must be identified and conveyed to all participants. Adherence to this protocol:
  - minimizes the occurrence of ineffective, “ad-hoc” meetings, and
  - sets preparation expectations for all session participants.

4.2 Problem Identification and Resolution

Because of the complexity of the interaction during the requirements generation process we cannot hope (or even expect) to provide a constraining structure that addresses all potential problems. As an initial step we have ascertained representative procedures and methods that help identify and resolve problems that still occur within the constraints of the protocol-imposed structure.

Problem Identification

Our experience and research have shown that, even within a structured environment, problems can still surface and, before being recognized as such, have a detrimental impact on the requirements generation process. Hence, enabling the recognition and identification of problems *as they occur* is an important part of our methodology. We achieve this by providing the customer and requirements engineer with procedures that help to identify problems quickly. The procedures reveal problems through sets of questions that, as part of the process, should be asked periodically. Examples of such questions include:

- Have "unnatural" constraints been placed on requirements?
- Are participant interruptions having an adverse impact on requirements?

Associated with each question are a set of indicators that enable one to accurately measure and answer the question.

Problem Resolution

Problem identification is only the first step in improving the requirements generation process; the second step is problem resolution. Based on problem characteristics, procedures direct the customer and/or requirements engineer toward one or more methods that help resolve the identified problem. These methods consist of specific tasks enabling the resolution of problems. For example the "pre-conceived mindset" presents a problem in which either the customer or requirements engineer unduly constrains the requirements generation process by focusing exclusively on a single “Solution” approach. Using procedures and methods we identify and resolve this problem by:

1) Asking the question, “Are pre-conceived ideas constraining requirements generation?”
2) Recognizing the indicator: *Requirements are reflecting design characteristics*, and
3) Applying the Method: *Enforce an initial identification of multiple alternatives to solving a problem*.

5. Summary and Work in Progress

A successful software development effort is one that produces a quality product within budget and on time. An integral part of such an effort is a set of well-defined processes that structure and guide development activities within an appropriate framework. Because product quality is a direct reflection of how well it meets the customer’s needs, the ability to capture customer requirements accurately and succinctly is paramount.

In this paper we have presented a refined model of the requirements generation process that promotes the accurate capturing of requirements. The major components of that model are:

(a) a framework reflecting a refinement (or decomposition) of the requirements generation phase into mini-phases – each mini-phase has its own unique set of objectives and activities associated with it,

(b) guidelines and protocols designed to structure the interaction process and to guide the customer and requirements engineer in their discussions, and

(c) a methodology consisting of protocols, procedures and methods to address potential problems an to identify and resolve existing problems.

Work in Progress

The work identified in this paper is "work in progress." What is being proposed is a set of ideas and concepts that are constantly evolving. Clearly much work is still remaining. Our current effort is focused on:

(1) developing a better understanding and details of the framework, guidelines and protocols,

(2) identifying additional problem categories and refining existing ones,

(3) continuing to develop the methodology, i.e., the procedures and methods,

(4) determining how to

- develop an adequate USRL and the contextual information, and
- form functional grouping based on functional cohesiveness, and

(5) identifying steps to transform the USRL and associated contextual information into an SRS.

Finally, we need to apply our proposed model to an actual requirements engineering effort to substantiate and/or validate our approach.

References

[1] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1991.

[2] W. Perry, *Effective methods for software testing*. New York, NY: John Wiley & Sons, Inc., 1995.

[3] J. L. Connel and L. B. Shafer, *Structured Rapid Prototyping: An evolutionary approach to software development*. Englewood Cliffs, New Jersey: Yourdon Press, Prentice Hall Building, 1989.

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

[5] D. P. Freedman and G. M. Weinberg, *Handbook of walkthroughs, inspections, and technical reviews: evaluating programs, projects, and products*, 3rd ed. New York, NY: Dorset House Publishing Co., Inc., 1990.

[6] 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.

[7] 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.

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

[9] 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.

[10] P. Mambrey and B. Schmidt-Belz, "Systems Designers and Users: Fictions and Facts," in *Systems Design for, with, and by the user*, U. Briefs, C. Ciborra, and L. Schneider, Eds.: North-Holland Publishing Company, 1983, pp. 61-69.

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

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

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

[14] M. E. Fagan, “Inspection Software Design and Code,” *Datamation*, 1977.

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

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