A New Approach for Software Requirements Elicitation

Prasad Rajagopal¹, Roger Lee¹, Thomas Ahlswede¹, Chia-Chu Chiang², Dale Karolak³

¹ Department of Computer Science, Central Michigan University, U.S.A.
{lee, ahlswede}@cps.cmich.edu
² Department of Computer Science, University of Arkansas- Little Rock, U.S.A.
cxchiang@ualr.edu
³ Intier Automotive Closures, U.S.A.
Dale_karolak@yahoo.com

Abstract

Requirements elicitation is both the hardest and most critical part of software development, since errors at this beginning stage propagate through the development process and are the hardest to repair later. This paper proposes an improved process for requirements elicitation. The key improvements are: (1) to train the non-technical stakeholders (primarily the users) in the capabilities and limitations of computer hardware, software, and of software developers; (2) identify keywords while interviewing the stakeholders, visually as well as in text form; (3) use keyword mapping to generate candidate system requirements; (4) apply the techniques of Quality Function Deployment (QFD) and the Capability Maturity Model (CMM) during the elicitation process.

1. Introduction

This paper proposes an improved process for software requirements elicitation. The hardest single part of building a software system is deciding what to build. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later [7]. Therefore requirements elicitation, the first phase of the software development process, is arguably the most critical.

Software requirement has been defined by IEEE [12] as,

(1) a condition or capability needed by a user to solve a problem or achieve an objective; (2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document; (3) a documented representation of a condition or capability as in (1) or (2).

Requirements are not limited to the functionality of the system, as often supposed, but include other aspects. Different authors have presented different definitions, but there are clearly nonfunctional requirements as well as functional ones. Davis [6] classifies requirements as:

- Functional requirements
- Nonfunctional requirements
- Performance/reliability
- Interfaces
- Design constraints

2. Problems in Requirements Elicitation

Errors in requirements elicitation are, overall, the most serious in software development, and the hardest to repair. Studies by Beichter [1] (Figure 1) indicate that

Figure 1. Breakdown of system errors (Beichter)

70% of the systems errors are due to inadequate system specification and 30% of the system errors are due to design issues.
The SEI National Software Capacity Study [17] (Figure 2) indicates some major factors for system development failure:

<table>
<thead>
<tr>
<th></th>
<th>Very Serious</th>
<th>Serious</th>
<th>Nearly Serious</th>
<th>Not Serious</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inadequate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2. Breakdown of system errors (SEI)**

We see that inadequate system specification, lack of user input, and changing requirements are major factors that contribute to the failure of system development.

### 2.1. Classification of Elicitation problems

McDermid [14] gives a list of common elicitation problems, which can be classified as:

- **Problems of scope.** The boundary of the system is ill-defined, so that unnecessary design information may be given, or necessary design information left out.
- **Problems of understanding.** Users have incomplete understanding of their needs; analysts have poor knowledge of the problem domain; user and analyst speak different languages (literally or figuratively); “obvious” information may be omitted; different users may have conflicting needs or perceptions of their needs; requirements are often vaguely expressed, e.g., “user friendly” or “robust”.
- **Problems of volatility.** Requirements evolve over time, either because of changing needs or because of changing perceptions by the stakeholders.

### 2.2. Problems of scope

Requirements elicitation must begin with an organizational and context analysis to determine the boundary of the target system as well as the objectives of the system.

Less ambitious elicitation techniques, not fully addressing this concern, run the risk of producing requirements which are incomplete and potentially unusable, because they do not adhere to the user’s or organization’s true goals for the system. Performing an organizational and context analysis allows these goals to be captured, and then later used to verify that the requirements are indeed usable and correct.

Elicitation techniques can be overambitious as well. Elicitation must focus on the creation of requirements and not design activities in order to adequately address users’ concerns. Elicitation strategies which produce requirements in the form of high level designs run the risk of creating requirements which are ambiguous to the user community. These requirements may not be verifiable by the users because they do not adequately understand the design language. Also, requirements expressed as a design are much more likely to incorporate additional decisions not reflecting user or sponsor needs, i.e., the requirements will not be precise and necessary.

### 2.3. Problems of Understanding

A Savant Institute study found that “56% of errors in installed systems were due to poor communication between user and analyst in defining requirements and that these types of errors were the most expensive to correct using up to 82% of available staff time” [3]. Problems of understanding during elicitation can lead to requirements which are ambiguous, incomplete, inconsistent, and even incorrect because they do not address the requirements elicitation stakeholders’ true needs. Lack of user input arises when users are not fully aware of their needs or are unable to communicate them. It also arises when analysts and developers fail to ask the necessary questions.

When a system needs to be defined, a series of meeting needs to be held consisting of stakeholders. These stakeholders include clients, users, software engineers, system analysts, domain experts, managers etc., Its been assumed that having more number of people in a meeting helps refining the system requirements and brainstorming becomes much effective and easier. But there is one potential problem having more stakeholders in a meeting. The language barrier is considered to be a major problem. When there is no proper common protocol to communicate the whole purpose of meeting together is defeated.

Different stakeholders may speak literally different languages, e.g. Chinese and English. But even within the same language, it is notorious that stakeholders from different domains (such as management, manufacturing, marketing, and technical) use the same words with different meanings. When literally different languages are used, there is the additional task of translating the relevant documents. When figuratively different “languages” are used, the problem may not even be recognized.
2.4. Problems of Volatility

One primary cause of requirements volatility is that “user needs evolve over time” [3]. The requirements engineering process of eliciting, specifying, and validating should not be executed only once during system development, but rather should be returned to so that the requirements can reflect the new knowledge gained during specification, validation, and subsequent activities. A requirements engineering methodology should be iterative in nature, “so that solutions can be reworked in the light of increased knowledge” [5].

Another cause of requirements volatility is that the requirements are the product of the contributions of many individuals, and these individuals often have conflicting needs and goals. For example, there is often more than one customer, with each customer having different and often contradictory views and interests [9].

Volatility also arises when clients or customers do not fully understand the capabilities and limits of the technology being offered. They often have unrealistic expectations of either the functionality that can be provided, or of the time scale in which the system can be developed. If these expectations are not corrected as early as possible in the elicitation process, the specification will incorporate them and will have to be revised later, at considerable cost.

3. Method Overview

The elicitation method proposed in this paper starts by dealing with the problem of lack of user input. The solution involves a series of sessions where users or other stakeholders are exposed to some of the basic idea of what they can expect from the developers and domain experts. They are also taught about the powers and limitations of the computer, and about the availability of other resources. This knowledge helps the users to clarify their needs and develop realistic rather than purely imaginative expectations.

Another problem is to define precise system requirements. Precision here depends on the identification and definition of keywords that exactly reflect the user’s needs and wants. Defining a system requirement is one of the hardest parts in any requirements elicitation process. If we get the system requirements wrong, we get the whole system wrong, and building a wrong system makes no sense. So the critical goal of any requirements elicitation is to build meaningful, precise and realistic requirements that reflect the needs of the user.

One of the main goals of this paper is to bring forth a new approach for eliciting good system requirements. The proposed approach starts by conducting a series of interviews, structured or unstructured, with various stakeholders. With these interview sessions we record all the keywords used by the participants. They are also given a template document, with which keywords can be classified for future analysis. With the templates filled out by the users and other stakeholders, along with the recorded keywords, domain experts can analyze each keyword and its meaning contextually. Here, a more important thing is to find out in what context each keyword is uttered. If we lose the context, the meaning of the keyword may be lost, and hence we fail to build a good system requirement specification.

Once we find the exact contexts in which the keywords are uttered, the approach maps each keyword to every other related keyword. Keywords that are related one way or another are grouped into sets. Each set of keywords can be diagrammatically represented as will be shown in section 4. With these sets prepared we further analyze each of the keywords and its elements to look for other elements in different sets for a relation. This process is repeated until the keywords are partitioned into nonintersecting sets.

3.1 Stages of Elicitation

The entire elicitation process can be divided into 11 stages. In this section we describe what each stage means and what elicitation problems it deals with.

Stage 1: Collect information about user needs and expectations

Lack of user input and unrealistic user expectations have long been considered very serious problems in the requirements elicitation process. This stage addresses these problems by compiling information about:

- Stakeholders (developers, software engineers and domain experts) abilities and domain knowledge.
- Limitations of computer resources and functionality, and availability of other resources.

This careful compilation of information will be used in the next stage to train the clients/users and make them aware of what they can and cannot expect from the software developers.

Responsibility: all stakeholders.

Stage 2: Train the stakeholders

Responsibility: developers.

Training the clients, users and managers with the information collected or compiled in the previous stage.
makes the users aware of what they can expect from
the developers or software engineers involved in the
project. At this stage, missing user input can be
supplemented and unrealistic expectations of
functionality or time scale can be weeded out.

Stage 3: Write descriptions of user needs
Responsibility: all stakeholders.
Each stakeholder involved will write a description
of his/her needs for the proposed system and of the
exact purpose of the system as the stakeholder
understands it. Since the clients and customers are
already educated about the computer limitations and
availability of resources through the training sessions
in the previous stage, they will have fewer, if any,
unrealistic expectations about a system they are
proposing to build. Realistic expectations at this stage
will also reduce volatility, since expectations are less
likely to change as the realities of the development
process become clearer.

Stage 4: Conduct oral interviews with users
Responsibility: developers.
The interviews at this stage are based on the users’
written descriptions from stage 3, and may be
structured or unstructured. The purposes of the
interviews are (1) to elaborate and refine the needs and
expectations expressed at stage 3, and (2) to identify
keywords used by the users, which will be critical to
the creation of formal requirements.

Stage 5: Map keywords
Well-defined operational definitions are essential
for building unambiguous system requirements for any
system. In this stage, the keywords identified in stage 4
will be analyzed and grouped by keyword mapping in
order to frame operational definitions. The keyword
mapping technique will be described in section 4.

Stage 6: Classify and prioritize system
requirements
After creating well-defined operational definitions,
we have a base for building clear and unambiguous
system requirements, and it becomes necessary to
prioritize and classify each requirement. The
classification is usually based on the project’s cost and
schedule.
At this stage, we also resolve conflicting
expectations among stakeholders, as well as any
remaining ambiguity or lack of clarity in the system
requirements as produced in stage 5.
We propose using Quality Function Deployment
(QFD), a requirements process technique, for this
purpose. The use of QFD will be described in section
7.

Stage 7: Fit the requirements to the domain
Forming domain specific requirements has always
been a difficult task and this depends on domain
experts and knowledge experts. Though the system
requirements formed in the previous stage are specific
and unambiguous, they may address issues outside the
problem domain — which are unnecessary, wasteful,
and may hinder the rest of the development process.
These issues can be solved only by employing
knowledge experts and domain experts. This task is
done in this stage.

Stage 8: Prototype
At this point we believe we have a correct set of
requirements. We test them by building a prototype.
Are these requirements complete, sufficient and not
excessive? What will be the outcome? Will this set of
requirements fulfill the needs of clients and other
stakeholders?

Stage 9: Check the prototype
Every prototype built should be checked for its
quality; in other words, every system requirement
should be tested for its realism, quality, unambiguity,
and exactness. With this quality check (repeated as
necessary) stakeholders have the opportunity to form
even more precise system requirements by eliminating
unnecessary information. This will further help
developers and software engineers to understand what
they are going to build and whether they can fulfill the
requirements of the clients.

Stage 10: Analyze risks and costs
This stage calculates and eliminates unnecessary
risks and costs. We propose to use the popular
Capability Maturity Model for its analysis. A brief
introduction will be given in section 8 about CMM and
how its features are used here.

Stage 11: Overall analysis
Finally we analyze the whole set of system
requirements to make sure everything done so far is
correct. From this point on, we actually start building
the system with careful supervision at each stage of its
development process.
4. New Techniques for Requirements Elicitation

The approach proposed in the paper presents a standard template consisting of problems in system specifications and flow diagram to solve that problem systematically and methodically.

4.1. Potential Problems in the previous model

Traditional requirements elicitation does not clearly explain any of its techniques or employ any of the early software requirements detection techniques available to avoid volatility, a major problem in the elicitation process. Elicitation starts with a data requirement phase which employs structured or unstructured interviews and brainstorming. But this often leads to lack of user input, or unrealistic user input due to poor understanding of resources and time constraints – even though the interviews are structured and developers are prepared to know what they want to ask.

One other major problem in requirements elicitation is that when all stakeholders meet together and start interviewing each other, language differences can be a major hindrance. This may lead to misunderstanding between parties, missing obvious information and conflicting views. When data acquisition is not done properly, developers will not know the exact requirements or may misunderstand the requirements, which in the future has to be changed; the problem of volatility involves huge cost.

4.2. New techniques to solve these problems

The proposed method addresses each of the above shortcomings. The following subsection explains how we address and solve each issue.

4.2.1. Training sessions to eliminate “lack of user input” and “poor understanding”

To avoid the problems of “lack of user input” and poor user understanding, at the beginning of requirements elicitation, stakeholders need to be trained or informed about the developers’ skills, computational abilities, the environment under which developers and other stakeholders are going to work, and what developers can offer to the customers. Most importantly, stakeholders need to be taught about what software engineers cannot offer in the project. This greatly reduces the problem of unrealistic expectations concerning functionality and deadlines.

This, in turn, reduces volatility of requirements at the beginning stage. Wrong or unrealistic expectation and unrealistic requirements may later on cause the whole system development process to fail.

4.2.2. Recording keywords

Many system development failures occur because the users cannot define their needs precisely, or because developers and domain experts miss “obvious” words that contribute essentially to system requirements. These problems can be avoided largely by recording each keyword spoken by each stakeholder.

Stakeholders will also list keywords on a template form with different columns for each class of keywords, e.g.: Behavioral keyword, Functional keyword, Non-functional keyword, purpose keywords, etc. Comparing the templates will show which keyword is mostly uttered by whom and hence identify their interest. If the same keyword appears in different columns, this is one signal of an ambiguity which will have to be resolved.

4.2.3. Pictorial representation of needs and wants to reduce language barriers

The usual procedure in multilingual situations is to employ a translator and translate each time somebody communicates with a new language. But whether or not more than one language is used, the problem remains that stakeholders from various domains (technical, management, marketing, manufacturing, etc.) understand words in different ways. Thus, even after we have identified keywords, their meaning is still often ambiguous or vague.

The resulting confusion can be greatly reduced by supplementing the keywords with visual images. Images or pictures speak more powerfully than words. Anybody from any part of the world will definitely understand images no matter what language he speaks and no matter how well he is educated. The process of “labeling” each keyword with a picture gives the stakeholders an opportunity to come to agreement about its meaning, and thereafter provides a convenient reminder of the meaning that has been agreed on. The Advantages of using this process are:

- Saves time on translating each document written in different languages.
- Agreeing on a common pictorial representation avoids conflicting interpretations of the representation.
4.2.4. Keyword Mapping

In addition to forming operational definitions, keyword mapping technique will largely avoid the problem of missing “obvious” information, which later may blow up to a stage where system development has to be stopped.

4.2.5. Operational definition extraction

Based on the keywords used by the stakeholders, domain experts can extract all those keywords that are specific to the domain. Finally form an operational definitions based on them. Once these definitions about the wants and needs are framed carefully, developers know what they should do which by implementing the well-defined specifications will satisfy the customers/users.

The details of keyword mapping and the use of keyword mapping to build operational definitions will be described in the next section.

5. Building operational definitions based on keyword mapping

From the customers and other stakeholders’ interview sessions we have recorded their wishes. From this a large amount of keywords are carefully analyzed, collected and organized. These keywords by themselves do not explain or tell us anything. But if we can relate them to each other, we are at least in a better position to define the users’ needs with the precision we need.

Starting with the templates filled out by the stakeholders, the keywords are organized into sets according to their type: functional, nonfunctional, performance/reliability, interface, design constraint. Some additional categories are desirable as well: behavioral keywords, which describe user initiatives and responses to the system; and attributes or desired properties of the system or parts of the system.

The technique is one-to-one mapping. With all the keywords recorded and represented in sets, we can capture the most obvious things that may be missed if not recorded and represented this way.

To do this, the paper applies a technique called keyword mapping (Figure 3). The technique represents each key word as an object or an entity. So each keyword represented as an object should have its own attributes and behaviors.

We use these attributes and behaviors to relate them with other keywords. Related keywords, say k1 and k2, form a new set which may be labeled S1. Similarly, a new set S2 is formed when a keyword k3 and keyword k4 are related. The key aspect of the technique is a keyword k1 is associated with a keyword k2 only if they are related in a specific way such as,

- Cost
- Design
- Knowledge domain
- Software issues

Or any of many others. Once we are left with no more keywords to form groups, we have each set $S_k$ containing only keywords that will form a well-defined requirement for a particular customer or stakeholder. The requirement elicited using this technique eliminates unnecessary information and is thus will be accurate and well defined, leaving software developers a clear map of what they have to do.

Each set can be represented as:

**Attribute Set** = \{(k1), (k2), (k3), (k4), (k6), (k7)\}

**Behavioral Keyword set** = \{(k1), (k2), (k3), (k4)\}

**Non-functional Keyword set** = \{(k1), (k2), (k3), (k4)\}

**Functional Keyword set** = \{(k1), (k2), (k3), (k4)\}

![Figure 3. Keyword Mapping](image-url)
Mapped Keywords representation

Map(Att, Behav) = {(k2,k3), (k4,k2)}

Map(Non-funct, Att) = {(k4,k3) , (k3,k7)}

Mapping Nested Sets

Figure 4 shows the next step in the process: mapping nested sets. Map(Map(Attr, Behav), Map (Non-funct, Attr)) = { [(k2,k3), (k3,k7)], [(k4,k2), (k4,k3)] }

Figure 4. Mapping Nested Sets

Now the keywords k2,k3,k3,k7,k4,k2,k4,k3 form a sentence. Thanks to the previous precautions of training the stakeholders and providing pictorial labels for the keywords, the sentence formed will be a system requirement that has been elicited in a systematic and neat way with

- No obvious information omitted
- No conflicting views, Ambiguity.
- No ill-defined system scopes
- No unnecessary information

6. Process Flow

The process flow diagram shown in Figure 5 is another representation of the elicitation process, slightly different from the 11-stage process described earlier, but more convenient for the remaining discussion.

Figure 5. Method Flow
7. Requirements Elicitation using QFD

7.1. What is QFD?

Quality Function Deployment (QFD) is defined by Soto [19] as “a systematic process for motivating a business to focus on its customers.” It is based on market research: understanding customers’ needs and desires, and the effectiveness of relevant products in meeting those needs and desires. In QFD, cross-functional teams identify and resolve the issues involved in developing products, etc., to satisfy their customers.

7.2. Why QFD?

Once a team has identified the customers’ wishes, QFD is used for two basic purposes [13]:

- To improve the communication of customer needs throughout the organization.
- To improve the completeness of specifications and to make them traceable directly to customer wants and needs. QFD requires that representatives of the different organizations involved in producing the product be involved in its definition. Consequently, these representatives discuss the meaning of the customers’ wishes and work together to ensure that they come to a common understanding. Communications throughout the organization are greatly improved. This process will also uncover many issues whose resolution will lead to a more complete specification.

QFD is organized around a model called the House of Quality (HOQ), a set of “rooms” encapsulating the various processes necessary to develop a complete and satisfactory product specification. The HOQ is built by in-house teams from various disciplines, under the guidance of a trained QFD facilitator.

As mentioned before, a prerequisite for QFD is market research, and the results of market research are the raw data used to build the HOQ. The QFD team then builds the HOQ, room by room.

Given one or more specific objectives (e.g., a narrow focus such as “optimize engine performance” or a more global focus such as “optimize overall passenger comfort”), the QFD process starts with obtaining customer requirements through market research. These research results are inputs into the House of Quality. The following is a discussion of each of the “rooms” of the House of Quality and how they are built.

**Figure 6. Components of QFD [20]**

The “Whats” Room: This room contains the requirements, as identified by the QFD team. “Typically there are many customer requirements, but using a technique called affinity diagramming, the team distills these many requirements into the 20 or 30 most important needs.” In affinity diagramming, the team discusses the initial requirements provided by the users, and clusters them into a smaller number of more general requirements. (This is an appropriate place to apply pictorial labels to keywords, as discussed in section 4.2.3.) The results are placed into the “Whats” room.

The Importance Ratings and Customer Competitive Assessment Rooms: When QFD is to be used, market research has to be designed around the expectation that the QFD team will use it. Market research provides information about the varying priority of expressed customer needs (the Importance Ratings room) and about the strengths and weaknesses of both the client’s and competitors’ existing products. Note that the Importance Ratings room is associated with the “Whats” room, while the Customer Competitive Assessment room feeds separately into the HOQ’s relationship matrix.

The “Hows” Room: The “Hows” room requires completion of the “Whats” room. In the “Hows” phase, the team develops metrics for success in the “Whats” previously identified. Each “What” requires
at least one “How”, and some “Whats” may require more than one.

**The Relationships Matrix Room:** After completion of the “Whats”, “Hows”, and Customer Competitive Assessment rooms, it is possible to start building the Relationships Matrix. The team attempts to define the relationship of every what to every How. The relationship may be strong, medium or weak, or there may be no relationship at all. In any event, the matrix must be completed.

**The Absolute Score and Relative Score Rooms:** In the Absolute Score and Relative Score rooms, the team (1) “creates a model or hypothesis as to how product performance contributes to customer satisfaction”, and (2) uses the Relationships Matrix and the Customer Importance Ratings to rate the various performance measures (the “Hows”) by their importance to customer satisfaction.

In moving from the Whats Room to the Score rooms, the technical members of the team play an increasing role, and predominate in the remaining rooms, though input from all team members is still essential.

**The Correlation Matrix Room:** We have noted that users’ expressed requirements often conflict with each other. When this happens, the conflicts are apparent in the “How” metrics. The Correlation Matrix room is where they can be resolved.

To cite an example from Squires:

“...Perhaps the customer wants a car that is fast, so your team comes up with the “how” of “elapsed time in the quarter mile”. After comparing performance between your car and the competitor's vehicle, you realize that “you blew the doors off the competitor's old crate”. However when you look in the Customer Competitive Assessment room, you see that most of the marketplace perceives the competitor's car as being faster. While you might have chosen one of the correct “hows” to measure performance, it is clear that your single “how” does not completely reflect performance needed to make your car appear faster.” [20]

**The Target Values Room:** At this point, the requirements have been identified, evaluated and tested in the preceding rooms. The final set of recommended specifications is placed in the Target Values room.

The preceding discussion of QFD has focused on eliciting requirements and developing a product specification. In QFD, this is often called the “phase one” matrix. A similar “phase two” matrix can be applied in the design phase, and even a “phase three” matrix during implementation or manufacturing.

As this description makes clear, QFD is a suitable structured process for developing system requirements for almost any product, including software. However, software development is distinctive in many ways, especially those discussed earlier in Section 2. Adapting the QFD procedure to the specific issues of software requirements elicitation and analysis is complex and we are still studying how this can be done.

### 8. Risk Analysis Using CMM

The following description is based on Fox [10]. The Capability Maturity Model for Risk Management is a model for describing both the present maturity of risk management processes in an organization, and for studying those processes in order to develop a more mature, i.e., effective, risk management process. The CMM is divided into five maturity levels [10]:

1. **Initial.** The decision support process for managing risk is characterized as ad hoc, and occasionally even chaotic. Remove an individual and the processes may change dramatically. Metrics may accurately measure factors which are significant for customer satisfaction, but customer perception may not match the technical criteria. Comparison of results from the Technical Competitive Assessment and Customer Competitive Assessment rooms can reveal such problems. Another example from Squires:

   “...Perhaps the customer wants a car that is fast, so your team comes up with the “how” of “elapsed time in the quarter mile”. After comparing performance between your car and the competitor's vehicle, you realize that “you blew the doors off the competitor's old crate”. However when you look in the Customer Competitive Assessment room, you see that most of the marketplace perceives the competitor's car as being faster. While you might have chosen one of the correct “hows” to measure performance, it is clear that your single “how” does not completely reflect performance needed to make your car appear faster.” [20]
commiserate with the next individual’s level of ability and experience.

2. **Repeatable.** Basic management processes are established to document the management of the organisation. The necessary process discipline is in place to repeat earlier successes on similar tasks, based on previous experience of the organisation.

3. **Defined.** The process for standardising, documenting, integrating risk management into the normal decision-making processes of the organisation.

4. **Managed.** Detailed measures of management decisions made, the formal process of managing risk and the quality of the risk management (planning, including setting context, risk identification, assessment of risk, evaluation of risk, mitigation of risk to an acceptable, the monitoring of risk and review of the whole process). All the business processes and the output products or services are quantitatively understood and controlled.

5. **Optimizing.** Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies to measurement of the public affected by the decision/s.

All levels except level 1 are analyzed into components which are used to develop strategies for improved risk management. At level 2, the task is to study the existing risk management process. At level 3, the goal is to work toward a “culture of effective business decision-making processes”, including strategic planning, plans for each unit, corporate education, process integration and development, and effective communication. At level 4, the goal is to develop quantitative measures of the organization’s risks, and at level 5, to implement a continual, measurable process improvement. Each level builds on the previous levels.

9. **Evaluation Methods**

We propose in this paper an effective data collection method for evaluating software development methodologies, from definition of the objectives of the data collection to analysis of the results. Any software industry is interested in the analysis of techniques, their integration into a new methodology, and the engineering of that methodology to particular environments [22]. An effective way to evaluate a methodology, understand the environment and refine the methodology for the environment is to collect data that characterizes the methodology and the environment and supplies insight into both.

The proposed method will therefore be evaluated by collecting data from domain experts and from other users who have studied the technique thoroughly. Data showing where changes were made, what kinds of changes were made, and the effort involved in making changes can be used to evaluate methodologies, characterize environments and permit the proper engineering of the methodologies for the environments.

Users will first be asked to evaluate the technique subjectively, by questions including the following:

- (true/false) In current software elicitation, many important things are overlooked.
- (true/false) Pictorial representations of keywords help to make their meanings clear and unambiguous.
- (true/false) The sentences generated by collecting, mapping and relating keywords can easily be converted to accurate system requirements.
- (true/false) The technique eliminates irrelevant, redundant or trivial requirements.
- (true/false) When many keywords (e.g. several hundred or thousand) are collected, the mapping process and its results are still manageable.
- Is this technique simple?
- Is this technique effective?

Next, the technique will be applied to actual elicitation situations of various scales, and stakeholders will be asked about their satisfaction with various parameters of the elicitation:

- Are the requirements complete?
- Are they within the scope of the system?
- Have irrelevant, redundant and trivial requirements been avoided?
- Do the requirements accurately represent the expressed needs of the users?

10. **Conclusion**

The goal of this proposal is to develop a new methodology for improved requirements elicitation. The major problems in most elicitation techniques derive from their imprecision, which leads to vague or even incorrect requirements. Specifically, we have proposed using,

- Training of users in the capabilities and limits of the computer and of software developers;
• Collection of keywords from stakeholders in all categories;
• Pictorial representation of keywords to facilitate agreement on their meaning;
• Keyword mapping to generate system requirements;
• Quality Function Deployment (QFD) to make sure that requirements are relevant to the task and to the users' needs;
• Capability Maturity Model (CMM) to make sure that requirements take into account the risks the system will encounter or generate.

With these improvements, we believe that software requirements elicitation can be raised to a new level of rigor and effectiveness.

References:


