Software Acquisition Meta-Model

Marc Mosko, Hong Jiang, Arindam Samanta,
Linda Werner

Jack Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

We present a software acquisition meta-model, called SAMM, for *Commercial-Off-The-Shelf* software acquisition. Our model is a meta-model in that it includes sections that use other, detailed, models for specific tasks. SAMM is a complete life-cycle model that begins with an end-user need and ends with software maintenance. We have adapted several other models in to SAMM and added several novel features that we think appropriate for acquisition of commercial software. Our model also addresses issues of process automation.

Contents

List of Figures

List of Tables

1. Introduction

This paper addresses the need for a practical software acquisition process that incorporates best practices of the software industry. We have developed an iterative model that focuses on user requirements and choosing software to meet those requirements. SAMM covers the complete acquisition life cycle, from initial user request to final software decommissioning. We based our model on requirements from Seagate Corporation.

Chapter 2 introduces our model with an overview of the steps and a process flowchart. The introduction also clarifies some terms used in the paper and notes any differences from [22]. Chapter 3 describes certain elements of the software acquisition life cycle that pertain to our model. The first stage of the SAMM model begins in Chapter 4, which concerns preparatory steps to start a model implementation. Chapter 5 describes our requirements gathering methods. It includes not only requirements elicitation techniques, but also requirements weightings. We use the weightings in subsequent sections to choose products. Chapter 6 describes our review process. Formal reviews happen at four stages in the model, with informal technical reviews more often. Chapter 6 presents the general framework for the review process and specifics for each of the four formal reviews. Chapter 7 describes our software selection analysis method. This stage of the model concerns the measuring of software features against requirements and making cost/time estimates for alternatives. Chapter 8 finishes the software selection process with suggestions for how to present analysis results to the customer. This section may either iterate the requirements/analysis process or continue in the life cycle model.

The subsequent sections of the model concern the Implementation phase. Chapter 9 presents several important aspects to purchasing software, such as terms of the agreement and code escrow. Chapter 10 concerns the modification and/or customization of the acquired software. The model may have determined in Chapter 5 and 7 that some form of modification or customization was needed to meet requirements. This section describes the aspects of such actions that would affect the acquisition model. Chapter 11 describes the portions of software pilot deployment that concern the acquisition model. Chapter 12 relates the general software deployment back to the acquisition model. Finally, Chapter 13 presents several aspects of on-going software maintenance that one must address to ensure that over time the acquired software continues to meet changing user needs. The paper concludes in Chapter 14 with our thoughts on the SAMM model and its application. We also provide a sample run through of the Choice Phase in Appendix A.

Some sections are, by necessity of space, very short. Their purpose in the present document is to call attention to a particular point and perhaps reference the related literature. Because we attempted to address the complete life cycle, the SAMM model has many stages.

2. Model Introduction

The SAMM model addresses the need for a practical software acquisition process that incorporates best practices of the software industry. We have developed an iterative model that focuses on user requirements and choosing software to meet those requirements.

In this introduction, we will first review a sampling of the literature on IT project implementation. From these case studies, we will extract key success components which should surface in our present model. We will then describe the general chapter format and define terms used in this study. The introduction then presents an overview of the SAMM model and the SAMM process flowchart. It concludes with a statement of Seagate Corporation's user requirements for the model and several design requirements introduced by the project team.

2.1   Model Motivation

There is a clear need for an acquisition model. Boehm *et al.* [7] note many risks associated with software acquisition. An organizational model can help a company successfully purchase and implement software. Such a model may help management conduct acquisitions and formalize user involvement.

In a study of 35 Korean companies, Shin *et al.* [40] found a positive correlation between management involvement in the acquisition process and the perceived quality of the software. This emphasizes the need for management to have a cohesive plan for software acquisitions that involves user groups.

In a meta-analysis of 25 software development projects, Hwang *et al.* [20] found a large correlation between user involvement and system success. They also found a moderate correlation between user participation and system success. The paper defines "user involvement as a need-based mental or psychological state of users toward a system and its development process and defines user participation as the observable behavior of users during the development process of a system." One could say that fostering a positive attitude toward the software system had a stronger correlation than actual physical tasks.

In a study of 5 south-east Asian companies that had been using IT technology for over 10 years, Jain [24] found that management involvement is important for successful IT projects. Managers in successful companies took an active role in fostering innovation through technology. Jain found three other constructs of a successful IT project: speed of diffusion of technology, managing quality, and sustainability. The study found that "IS professionals who were dispersed/distributed in functional areas created parallel groups of users [who accepted technology], thus facilitating widespread dissemination of technology." In terms of quality, successful organizations paid particular attention to user support and had a feedback mechanism such that users could review IT performance. Finally, the study found that successful organizations funded their IT departments such that they could sustain innovation and rapid diffusion.

From these case studies, we would deduce that a successful IT-driven process should have the following characteristics. Management must take an active, supportive role. They should demonstrate how new technology and innovation may benefit end-users. They should also support the IT group. Through user contact and efficient project execution, the IT group may bolster end-user enthusiasm for IT projects. Management should provide IT the

resources to engage users on a personal level. Finally, in the planning and implementation of IT projects, the project methodology should foster user buy-in and cultivate a positive user attitude towards IT projects. This should include user feedback channels.

2.2   Section Outline

Each of the following model chapters will state the *Purpose*, the *People*, and the *Procedure* for a given model stage. Each section will also state the *Inputs* and *Outputs* along with *Exit Criteria*. In most chapters, we also have a section entitled *Use of Database*. This organization is based loosely on [21].

Each chapter's Purpose section provides an overview of that section. It will state the Inputs of the chapter. The Inputs are documents or other artifacts produced in earlier sections. The Purpose section will list the Chapter Outputs.

The People section lists the groups involved in the Procedures of the chapter. The People section states each group's role in accomplishing the chapter's Purpose.

The Procedure section describes what should be done to achieve the chapter's Purpose. The Procedure section may also describe how it should be done. Depending on the chapter material, there may be brief digressions to related theoretical or conceptual topics.

Following the Procedure section, the Database section mentions the main points for automation. It lists the data items we consider important to track for long-term purposes. One may, of course, automate and electronically track more information than our minimum recommendations.

Each chapter will conclude with the Exit Criteria. Exit Criteria are the necessary conditions to conclude work in a section.

2.3   Terms

Generally, we use the same definitions as [22]. We add a few definitions and refine two definitions. In this subsection only, we show defined terms in boldface.

1. **commercial-off-the-shelf (COTS)**: We modify this definition to include any software that may be changed by the **acquirer**. We thus include software packages with built-in macro capabilities or a scripting language. We also consider actions such as creating automatic installation scripts a potential **acquirer** activity. The **acquirer** may have the **vendor** or a third-party perform these activities.

2. **customer**: An individual or group who begins the acquisition process; also the entity that will fund the endeavor.

3. **end-user**: An individual, either in a **user group** or the **customer**, who will use the software.

4. **functional requirement**: see **user functional requirement**.

5. **IT group**: The technical division of the **acquirer** organization. The **IT group** is the target audience for this model.

6. **stake-holder**: Any group or individual with an interest in the software. This may include entities with an indirect interest. Generally, one would identify a stake-holder as a **user group** that would directly use the software or whose job functions the software would directly affect.

7. **modified-off-the-shelf (MOTS)**: a COTS package that requires **supplier** changes. These changes may only be done by the **supplier**. These would include changes to source code to add or change functionality. They may also include custom developed add-ons.

8. **non-functional requirement**: a characteristic of the software that may not be observed or tested in the software. These might include items such as vendor organization size or corporate strategic goals.

9. **system functional requirement**: a characteristic of the software observable by the **IT group**. These might be platform restrictions, memory capacity, or compliance with standards.

10. **technical requirement**: see **system functional requirement**.

11. **user functional requirement**: a characteristic of the software observable by **end users**. A behavioral specification.

12. **user group**: We view the IEEE **acquirer** as two entities, one (or more) user group and the technical **IT group**. The user group would generally be associated with a department, but it could be any convenient grouping of **end-users**. One could group users based on expected software usage rather than organizational hierarchy. A user group could also be external to the **acquirer**.

13. **vendor**: We use this term interchangeably with **supplier**.

2.4   Overview

We divide the software acquisition process in to two distinct phases. We call the first the "Choice Phase". The second is the "Implementation Phase". The purpose of the Choice Phase is to choose the COTS/MOTS software that most closely meets user requirements. This phase also makes estimates for a fully custom application if no such COTS/MOTS software were to be found. The Implementation Phase is mostly a process control phase. We oversee the implementation of the chosen software package to make sure it is implemented as anticipated and performs as expected.

We view the Choice Phase as iterative. The acquirer would perform as many rounds as the customer needed to pick a product. Generally, each round would refine requirements and perform any needed analysis. These results are presented to the customer who makes the final decision to buy or iterate. Based on a previous iteration's analysis and testing, the following requirements stage may modify or remove existing requirements, in a *capabilities-to-requirements* style [8]. [1]

The Implementation phase is sequential, possibly with some parallel activities. The SAMM model does not specify a deployment methodology, but it does state certain feedbacks that should flow from the deployment teams back to the acquisition group. The customer made a decision to buy a product based on certain representations, either by the vendor or by the IT group, that the product could be implemented for a certain cost and

---

[1]One may compare our model with the WinWin Spiral model [6]. Both models begin by identifying the stake-holders in the project. Our model proceeds to define requirements to a sufficient point that we may analyze competing implementations (COTS/MOTS software). With each iteration of our Choice Phase, we keep refining requirements and testing possible implementations, similar to the cyclic nature of the Spiral model. With each iteration of the Requirements stage, new stake-holder views may surface, similar to the explicit stake-holder consideration of the WinWin model.

within a certain schedule. In the Implementation Phase, the SAMM model observes cost, schedule, and satisfaction factors[2]. If any of these exceed a customer threshold, the acquirer should review the purchase decision and possibly implement remediations.

There are four stages to the Choice Phase. We also include two reviews. The stages are Preparation (Section 4), Requirements (Section 5), Analysis (Section 7), and Presentation (Section 8). The two review stages are Requirements Review and Analysis Review (both in Section 6).

In the Implementation Phase, we break down the acquisition process in to five stages. This phase is sequential, barring any major problems with the chosen software. The stages are: Purchase (Section 9), Modify/Customize (Section 10), Pilot Deployment (Section 11), General Deployment (Section 12), and Maintenance (Section 13). There are also two formal reviews in this phase, one after the Modify/Customize stage and one after the Pilot stage. We would note that sometimes the Modification/Customization activities may be pipelined with pilot group deployments and general deployments. By describing this phase as sequential, we do not mean to inhibit efficient deployment schemes.

The Implementation Phase also has two "Remediation" terminations in cases where significant problems arise with an acquired software package. The specific activities of these steps would come from a risk management plan. Such a plan should be considered and defined by a risk management group.

2.5 Model Flowchart

This section presents a process flow chart for the SAMM model. We wish to point out the main flow and four interfaces. Referring to Fig. 2.5, we will present an overview of the data interchange between process stages noted by the numbers 1, 2, 3, and 4. In particular, we wish to call attention to data items that should be traced through the model.

1. The Preparation stage will generate the initial acquisition project. This will include staffing, original problem request, estimated budget and desired time schedule. This stage may also document any potential software solutions known to the customer. It will also produce a list of stake-holders.

2. The Requirements stage will generate four documents: a weighted user requirements statement with use cases, a weighted technical requirements specification, a technical requirements quality estimate, and a test plan.

3. The Analysis section will produce several items.
   (a) A list of potential vendors.
   (b) An RFP that was sent to vendors.
   (c) The vendor responses to the RFP and an analysis of the responses.
   (d) A time/cost estimate for each vendor's solution. This may include time/cost estimates for customization and/or modification, as deemed necessary by the IT group or the vendor.
   (e) A time/cost estimate to develop the software from scratch.
   (f) Test results of vendor software. Using RFP responses and in-house testing, the Analysis stage will reduce the data to a weighted score for each user requirement.

---

[2][35] notes several other indicators for quality besides our common understanding of "satisfaction."
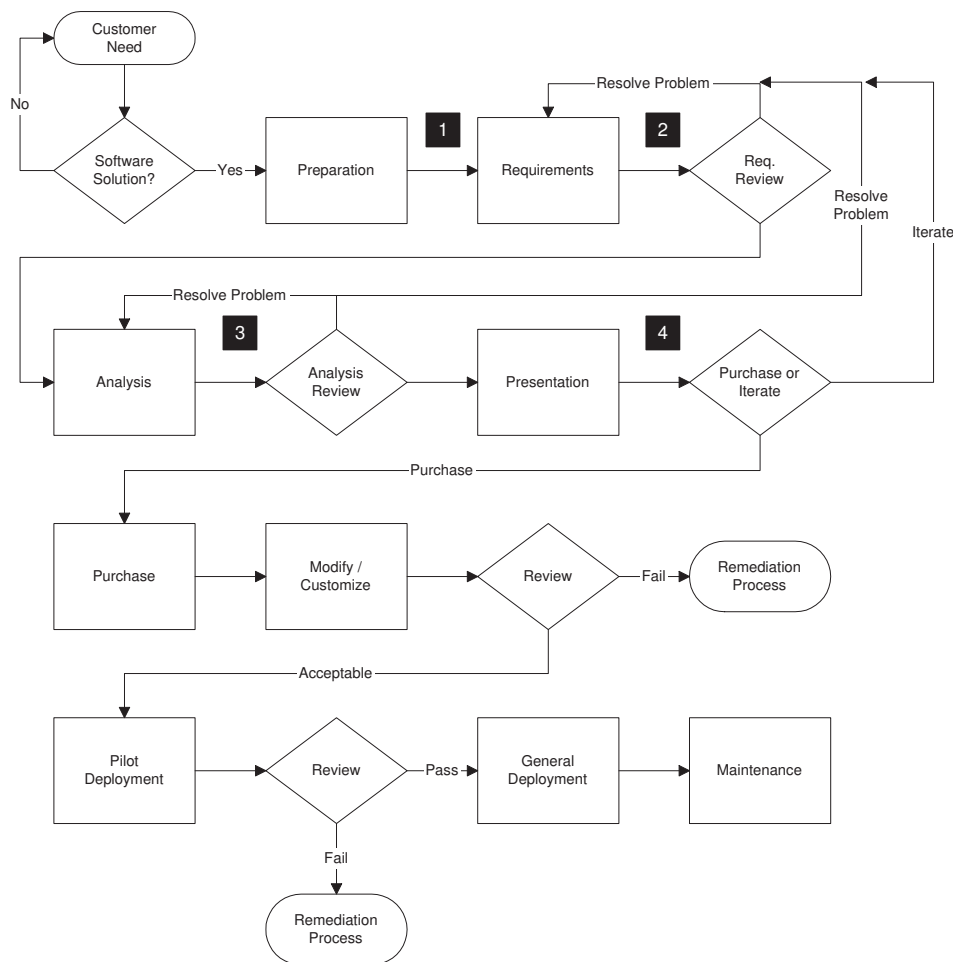
Figure 2.1: SAMM Process Flow

4. The Presentation stage will, among other things, produce a "buy" decision that is predicated on certain results from the analysis section. The Presentation section will document this reasoning.

2.6   Model Requirements

Tables 2.1 and 2.2 list, respectively, the Design and Seagate project requirements. Design requirements were introduced by the project team as, generally, project wide objectives. The Seagate requirements came from masterials provided by Seagate and an in-person interview with a key IT manager at Seagate. The Seagate table lists the Section number(s) where we address each requirement. The Design requirements are not section specific, but generally guided the project work.

Some Seagate requirements were addressed in a specific section. Some items were covered by an optional database. Such items are indicated with a section name of "DB". Some items are covered by the iterative nature of the Choice phase. These are indicated by a section name of "Iterative".

We determined that one requirement was unneeded. In our terminology (see Section 5.3.2), it is a "non-validated" requirement: it did not add to the product being built. It is noted by a section name of "unneeded".

Although requirement R-07.1 is touched upon in Section 5.3.1, it did not get the attention that it deserved. A future version of this process should address this deficiency.

Similarly, requirement R-18.1 does not receive sufficient attention. We use metrics throughout the process, but we do not specifically address relative metric value.

ID	Design Requirement
D-1	The process should have a general structure that relates the pieces.
D-2.1	The process should identify all groups potentially affected by an acquisition.
D-2.2	The process should explicitly consider global nature of company.
D-3.1	The process should clearly define who does what.
D-3.2	The process should be modular such that we may design it in a short time.
D-3.3	The process should have internal checks.
D-4	A planned database structure should support the decision model. See R-11, R-12.
D-5	The process should be usable by Seagate. Stress practical, less theoretical.

Table 2.1: Design Requirements

ID	Sections	Title	Description
R-01.1	14	Low Startup Cost	No more than $20k
R-02.1	14	Time span	Should take no more than several months for large acquisition.
R-03.1	14	Personnel: IT Group	1 Project Manager, plus other staff on an as-needed basis. Generally small groups work on projects.
R-04.1	DB	IT Department	Geographically diverse locations, effectively independent projects right now.
R-04.2	DB	IT Department	Need method to coordinate regions.
R-05.1	5	End-user	Small load as practical.
R-06.1	5.3.2, 6 7.3.2, 7.3.4	Ambiguous Reqs	Need detection method.
R-06.2	Iterative, 5.3.1	Ambiguous Reqs	Need resolution method.
R-06.3	7.3.4	Ambiguous Reqs	Need testing strategy to help consistency of results.
R-07.1	5.3.1	Templates	Development templates do not seem appropriate.
R-08.1	7.3.1	Cost Estimation	Does not have in-house cost estimation method for custom development.
R-09.1	unneeded	Decision Tool	Has in-house decision aid tool from defunct company.
R-10.1	5.3.1	Req. Elicitation	Would be interested in summary of pros/cons of different methods as applied to different user communities.
R-11.1	DB	Deployed Software	Need to know what is deployed.
R-11.2	DB	Deployed Software	How will changes affect it.
R-11.3	DB	Deployed Software	How to detect conflicts.
R-12.1	DB	Low automation	Databases should be incremental, low startup overhead.
R-13.1	9	Non-Technical	Plan should address code escrow.
R-13.2	7.3.2, 8	Non-Technical	Plan should address vendor aspects.
R-13.2.1	7.3.2, 8	Non-Technical	Vendor size: are they able to support large company?
R-13.2.2	7.3.2, 8	Non-Technical	Vendor size: are they financially stable?
R-14.1	5	Requirements	How to document requirements?
R-14.2	5	Requirements	How to prioritize and weight to find best fit?
R-15.1	3	SALC	Could the Acquisition Life Cycle be a variant of existing, in use life cycle?
R-16.1	7	Decision Support	How to make the build vs. buy decision.
R-16.2	10, 11	Decision Support	What would change one's mind?
R-17.1	6	Formal Review	What should be formally reviewed?
R-18.1	5, 7	Metrics	What are good ones and how should they be used?

Table 2.2: Seagate Requirements

3. Software Acquisition Life Cycle

3.1 Purpose

This section presents several overarching features of the software acquisition life cycle. These concepts should apply to all other sections of the model.

As we did not find a *acquisition* life cycle model in the literature, we present such a model in the present work.

3.2 People

1. **IT Group**: All sections.

3.3 Procedures

In this section, we call attention to certain aspects of the acquisition life cycle. These activities are done throughout the life cycle model. The main procedures are maintaining good documentation and tracing information through the acquisition process.

3.3.1 Planning for change

It is important to remember that user needs will change over time. The act of implementing a software product will, in itself, change the work environment. All parties involved with software acquisition will need to periodically review requirements and software products to ensure that the two still coincide.

3.3.2 Documentation and Traceability

Over the life of a software product, it is important that the IT group, in particular, maintain a written history of software products. There are many practical benefits. Over time, one main benefit will be for impact analysis. When the acquirer wishes to change systems at a later date, it becomes important to know how that will affect installed software. If software has dependencies on other software or specific operating system versions, for instance, even minor changes may have unforeseen consequences. See [46, 41] for a practical introduction to writing requirements documents. The topics in this section are based on [2].

We do not intend for these documentation and traceability requirements to be burdensome. The main objective is to create a paper trail (or e-trail as the case may be) such that one has forward and backward references.

Documentation should be maintained in a revision control system. This may be an automated system or a manual process. We would recommend the following minimum practices:

1. Control the releases of a document, and number each release (this could be by date, by version number, or any convenient notation).

2. Maintain a history of released documents.

3. Appoint a person responsible for each document.

4. Maintain a version change history that indicates what changed from version to version. This is not necessary for the first version.

5. Use change request forms. Any interested party (stake holder) may request a change to any part of the process. These change requests should be documented. Their approval/denial should likewise be documented. If approved, any documents affected by the change should indicate the source of the change (i.e. the change request).

Traceability means that at each stage of a project, one may determine from where one came and to where one will proceed. This is important so that when requirements change, one may find the affected pieces. If the customer drops a requirement, for instance, one should no longer test for that requirement and make a "buy" decision based on such tests. For a good, practical overview of traceability and sample implementations, see [41]. Specifically, in our model, one should trace:

1. Who asked for a user requirement and why.

2. Who developed each use case.

3. Which IT staff translated a user requirement in to technical requirements.

4. The bidirectional relationship between user requirements and technical requirements.

5. The bidirectional relationship between user requirements and use cases. This may be a multi-directional relationship if a use case does not span all related technical requirements.

6. The bidirectional relationship between test plan items and the use cases or technical requirements that generated them.

7. The association between testers, their scores, and test plan items.

8. The association between analysis time/cost estimates and implementation progress.

4. Preparation

4.1 Purpose

This stage defines the general or informal requirements of the software acquisition process as a preparation stage of the requirement stage. It begins upon an original user request. The IT group produces a project definition document.

4.2 People

1. **IT Group:** Involved in the preparation of both project and managerial definitions.
2. **User:** Involved in the preparation of project definitions only.

4.3 Procedure

This stage actually initiates the software acquisition process with informal definition of the total process. More specifically an informal definition of software acquisition can be categorized into two parts project and managerial definitions.

1. **Project Definitions:** Project definition sub stage defines the main constraints of software acquisition project such as time, budget, project type.
   (a) **Time:** Time defines the total time allocated for the acquisition process.
   (b) **Budget:** Budget defines the total money assigned for the process.
   (c) **Project Type:** In the project type, the project is classified into two categories simple and complex depending upon the project size. Simple project category skips some stages of the acquisition process making the acquisition process more efficient. Thus the project type also defines the subset of all software acquisition stages to be followed.
      i. **Simple Project:** A small budget project or a project with clear initial vendor choice is considered as a simple project. A simple project skips the formal review stage and defines vendor RFP sub-stage informally.
      ii. **Complex Project:** A complex project has poor initial requirements, high visibility, strategic importance and lastly expected difficult choice.
2. **Managerial Definitions:** From managerial point of view the preparation stage defines the review boards, the stake-holders and the pilot group of the project.
   (a) **Review Boards:** The managerial definitions define the review boards. A review board analyzes and approves all the decisions taken in successive stages of the cycle. It defines the review board members by selecting the members from IT group, user, management and possibly legal groups.
   (b) **Stake-holders:** Managerial definitions define the stake-holders of the project. The stake-holders of an acquisition process are the people who would be affected by the process.
   (c) **Pilot Group:** It likewise defines the pilot group for deployment.

4.4   Use of Database

This stage also defines the database which is used to gain experience from the past project definitions.

4.5   Exit

Exit from this stage takes place after the completion of a project file with all the above general and informal requirements, definitions listed on it.

5. Requirements

5.1   Purpose

The Requirements section takes the original problem description from Preparation and produces a testable, measurable plan that one may use in the Analysis section. The main activity in this stage is to determine, through various means, the functional and non-functional software requirements.

This section will produce the following documents:

1. **Weighted User Requirements**: The functional and non-functional user requirements. The user will assign a priority or weight to each requirement that indicates its importance to the user.

2. **Use Cases**: The user groups and IT group will develop use cases. Use cases are our main method for discovering user requirements, the translation of user to technical requirement, and the generation of test cases.

3. **Test Plan**: The IT Group, in conjunction with the user group, will devise a test plan for the user requirements. The test plan will be as exhaustive as reasonable, with priority given to important user requirements. The test plan may also include IT group technical requirements that do not directly correlate to a user requirement.

4. **Requirements Quality Measure**: The IT group will produce a document showing the relative quality of the user requirements. This document is an indicator of how well the acquirer understands the requirements. This document is used by the Requirements Review stage.

5. **Weighted Technical Requirements**: The IT group will translate the user requirements in to technical requirements. A technical requirement is a measurable/testable statement of a user requirement.

5.2   People

1. **IT Group**: All sections.
2. **Customer**: Requirements specification.
3. **Stake-holders**: Requirements specification.

5.3   Procedures

Four main sections comprise the requirements procedure: requirements elicitation, test plan, requirements quality, and technical requirements. Requirements elicitation is the process of ascertaining the user requirements. It includes a discussion on elicitation techniques and general requirements features. In the Quality section, we describe a method by which one may determine an approximate quality measure. We measusre quality as low ambiguity, completeness, and correctness. The Technical Requirements section describes how one may translate user requirements in to testable, technical requirements. Finally, the Test Plan section describes considerations in constructing a plan such that one may test for the presence of user requirements in potential COTS/MOTS software.

5.3.1   Requirements Elicitation

The goal of Requirements Elicitation is to build a set of user functional requirements. The user group will assign a weight to each requirement. The weight indicates the importance of the requirement to the user group. The IT group will then take the weighted requirements and translate them into system functional requirements, as needed. Through a testing process, each potential vendor product will receive a grade that is influenced directly by the user requirement weights.

In this section, we describe methods for requirements elicitation, social influences in requirements engineering, and issues with weighting. One may also see [37, 41].

Requirements Elicitation Methods

There are many methods by which one may elicit user requirements. We briefly describe several techniques before a more detailed description of our preferred method.

General Techniques   We begin with a summary of four techniques from Goguen *et al.* [16]. Goguen recognizes that "The problems of requirements elicitation cannot be solved in a purely technological way, because social context is much more crucial than in the programming, specification and design phases." We shall return to this observation several times in the material below. Of the several techniques in Goguen, we shall concentrate on *Interviews* and *Protocol Analysis*, as they are techniques commonly used to elicit requirements. We follow the Goguen's tone, which primarily points out the stumbling blocks to successful usage.

According to Goguen, there are three main types of interviews: questionnaires, open-ended interviews, and focus groups. Goguen notes that questionnaires, while appearing innocuous, generally have interpretation problems. The same questions may have different meanings to different respondents. In regards to the open-ended interview, the article advises that one should not ask people to describe activities that they normally do not describe. The example given is to ask someone to describe how to tie a shoe. This may be generalized to common work functions that the interviewee does not normally verbalize. When discussing focus groups, Goguen suggests that they may offer more spontaneous conversation than other interview techniques. However, respondents will be limited to exchanges concerning products or technologies the group understands. Focus groups would be less useful in discussing new technologies not understood by the group.

Protocol Analysis is the process of asking a subject to solve a problem or complete a task while describing their thought processes. An observer then records the actions and words. Goguen argues that protocol analysis does not work. The subjects will gear their discourse for the observer. The process of working and describing is an unnatural form.

For a formal introduction to methods of enquiry, one may look to, for instance, [15]. Rigorous designs would consider such aspects as time-evolution of subject responses. One may design interview questionnaires with specific types of cross-checks that facilitate data validation and control sample errors.

Use Cases   Our preferred requirements elicitation method is Use Cases. The following material on Use Cases is from Rumbaugh [36]. For a more complete tutorial on use cases, we would direct the reader to [38]. We believe Use Cases are appropriate for our model

because they (a) are user-centered, (b) proceed from general requirements to specific, (c) the system is a "black box" and only externally observable behavior is modeled, and (d) Use Cases are well received by practicing methodologists.

Use cases are built by an *actor* role-playing with the hypothetical system. An actor may be a person or an another automated system that will interact with the system in question. The actor focuses on particular sequences of actions that would accomplish a stated business goal. In some ways, this is similar to Protocol Analysis, except we are not concerned with the exact physical activities of the actor, nor the actors psychological reasoning.

Uses cases may be organized and associated. Rumbaugh describes two possible methods. For a COTS/MOTS software acquisition, we think that one does not need a high degree of use case decomposition. One may identify main-line cases, such as a hotel quest check-in. A main-line case may use "subroutines", such as asking "smoking or non-smoking." The subroutine does not stand alone. A main-line case may also use other main-line cases. During the check-in case, the clerk may ask "would you like to valet park your car?" The valet parking case could stand alone as a separate case. We think that such decomposition would have limited benefit for COTS/MOTS software, as one is not designing an implementation. Identifying the cases is important, but constructing detailed relationships and calling graphs would probably be excessive.

We do, however, believe that one should organize use cases along general work processes and also by data content. One could group use cases by the underlying data. End-users could then be identified by data and their use cases compared. As an example, we could look at a pension deduction option made by an employee. We could identify at least two end-users who use this data. Finance would need it to make adjustments to a paycheck and Human Resources might need it to make sure it meets a benefits package. Thus, one could group use cases by data, and find end-users who have potentially different viewpoints on the data. Candidate COTS/MOTS packages might need then to accommodate these uses.

Another benefit of use cases is that they may lead to *in situ* Test Plan cases. If users were to evaluate candidate COTS/MOTS software along the lines of their stated requirements by way of use cases, one could measure how effective a package is at meeting practical needs. Such testing may also show process re-engineering possibilities. In the reciprocal process of stating requirements and evaluating capabilities, use case testing might lead to specific process iimprovements. As users and the IT group test products, their understanding of software capabilities may change. These new views, expanded or contracted, may modify how users think about their own business process. They may also affect the stated requirements of use cases.

View Points   We will not give a technical definition of "viewpoints" due to space. We wish to note that in every-day usage different viewpoints exist in corporations. This is especially true between different business functional groups. Teo [44], for instance, finds in a study of 600 firms seven drivers of viewpoint divergence between executives in business planning and in IS departments.

Viewpoint requirements engineering essentially asks each end user or user group how they would use the software. Through a thorough analysis, viewpoint engineering tries to construct the total end user view of the problem by looking at each perspective.

Menzies *et al.* [31] finds that a thorough and extensive use of viewpoints has limited effect on knowledge content on requirements. Detailed probing of all possible user groups is not needed from a requirements point of view. They find that only if the stake-holder "worlds"

are truly different does a multiple viewpoint methodology add significant knowledge. The authors acknowledge that there may be several significant side benefits to using viewpoints methodology, such as stake-holder buy-in.

Non-functional Requirements   Several papers also emphasize that organizational or corporate objectives may be included in system requirements. Avison *et al.* [3] suggests that IS projects should capture both current needs and expected future uses. They present a "'vision process' ... by which strategic vision may be made operational." In a loosely related study, Teng *et al.* [43] found that in business process re-engineering, the long-term significant factors were not IT technical expertise, but the coupling of IS planning with business strategy and the overall innovative ability of the organization.

When gathering requirements, one should try to focus on the business goals, not necessarily the business process. Clark [11] studied 30 companies from the United States. In regards to using COTS/MOTS (called "external software packages" in the article), the authors found that "Most [managers] did agree, however, that the objective was to modify the software as little as possible and use of a strategy changing internal operations to fit the software to avoid modification. The majority used a rule of thumb that required the software to perform at least 80% of its intended function without modification." This suggests that companies are willing to change their processes to accommodate COTS/MOTS software.

Social Influences

Potts *et al.* [34] presents an interesting view on the process of requirements elicitation. The paper presents a technique known as *naturalistic inquiry*: "These techniques derive from two traditions: the anthropological tradition of ethnography and the more recent sociological tradition of ethnomethodology." The authors claim that "Using these types of data during design may lead to a system to which people can adapt and be productive when the system is installed, thereby increasing the likelihood that it will be used effectively."

The article calls attention to the divergence of Naturalistic Inquiry and traditional requirements engineering. Traditional RE "is founded on the philosophical tradition of positivism, which construes knowledge as accruing through the systematic observation of stable and knowable phenomena. Such a world view is consistent with the objectives of RE, because the activity of capturing requirements assumes that stable and specifiable phenomena are out there in the customer's world available for discovery through surveys, interviews, and the close reading of informal requirements documentation...."

We note here only that Naturalistic Inquiry generally is at odds with the positivist view. Potts lists five basic tenants of NI. Essentially, they state that one cannot objectively determine requirements, in the same detached way that one may observe physical phenomena. Inquiry is value laden, and subject's interpretation of reality is a construction of the subject's social context.

From a practical view, we would suggest that when conducting requirements engineering, the RE be aware of his or her tacit positivism. It may not always be possible to identify cause from effect or to abstract common patterns.

Weighting

User requirements should be weighted. If requirements are based on use cases, then use cases should be weighted. For a simple model, we would suggest a response scale of 1 to 5, where 1 is the lowest weight and 5 is the highest weight. One may wish to scale the range based on the number of requirements, such that the highest weight has a numeric value in proportion to the number of lower weight requirements. We discuss other issues below in Sections 5.3.4 and 5.3.5.

We recognize that users will sometimes have a difficult time choosing between importance levels. The number of levels should be kept low, even if their numeric values widely vary. Between iterations of the Choice Phase, requirement weights may change. A principle driver would be cost and capabilities. Cost and capabilities are essentially the same in this context. If a "5" requirement is only satisfied by a COTS package that costs three times more than the other packages, the end-users may re-evaluate the importance of the requirement. If no COTS package exhibits the capability, one must either find a MOTS package (generally more expensive) or undertake a full custom development (even more expensive). Requirement priorities that are set for political reasons may be particularly prone to this type of adjustment.

Schneider *et al.* [38] has a weighting system specific to Use Cases to estimate work in a project plan. Several of their ideas might be useful in developing user-preference weights. One may weight Actors such that activities which involve specific Actor(s) have proportional weights. One may also weight use cases based on the number of business transaction they contain.

IT system requirements

The IT group may add their own system functional requirements. These may be items such as training time, hardware requirements, or serviceability. Generally, these requirements would be used in computing the effective cost of a product and would not enter in to the actual weighted requirements. These requirements may also affect a product's delivery schedule. We would prefer that the "grade" of a product be based on user needs.

We expect that the IT group may develop a generally applicable template of IT concerns for software packages, such as supported platforms, installation procedures, and dependencies.

It is also incumbent on the IT group to evaluate the security parameters of COTS/MOTS software [30]. Such evaluation is outside the user scope. One must also beware of some vendor marketing claims. Products that handle sensitive corporate information or that connect to the Internet should undergo a risk assessment. The RFP, described in Section 7.3.2, could include specific questions on product security.

When are you done?

Wiegers [46] suggests the following six signs that one has reached the practical end of requirements elicitation:

1. Users cannot think of any more use cases.
2. Users propose new cases, but they contain the same functional requirements as previous cases.
3. Users begin to repeat issues already raised at previous meetings.

4. New use cases or functional requirements are all out of scope for the project.

5. New requirements are all low priority.

6. New requirements are future wishes, but not needed in the immediate software product.

5.3.2   Quality

Measuring the quality of the software acquisition requirements specification is an important stage in the software acquisition process. A measure of quality identifies several characteristics such as unambiguity, completeness, correctness, verifiability, conciseness [13]. We took the quality model described in [13] to perform a quality measurement on unambiguity, completeness and correctness of the software acquisition requirements.

1. Unambiguity: Unambiguity is an important characteristic of the software acquisition requirements. Less unambiguous requirements are easier to understand. Using the definition of [13] it can be said that a software acquisition requirements document is unambiguous "if and only if every requirement stated therein has only one possible interpretation". Thus an ambiguity of software acquisition requirement depends on the understanding power of a person. Reviewing is the best way to measure how ambiguous a requirement is. According the definition in [13] "an ambiguity is the percentage of requirements that have been interpreted in a unique manner by all its reviewers". Unambiguity is defined as Eq 5.1 [13].

$$Q_{unambiguity} = \frac{n_{ui}}{n_r} \qquad (5.1)$$

In the equation Eq 5.1 $Q_{unambiguity}$ represents the unambiguity of the software acquisition requirements, n_{ui} represents the number of requirements for which same interpretations are predicted by all the reviewers where n_r represents the total user requirement in the software acquisition process. Mathematically, $0 \geq Q_{unambiguity} \geq 1$ where a value 0 of unambiguity shows that all the acquisition requirements are confusing and similarly a value 1 of unambiguity shows that all the acquisition requirements are clear and easy to understand.
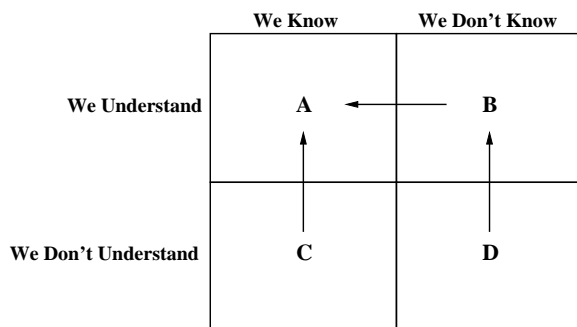

Figure 5.1: Alexander's Requirement Completeness Model [1]

2. Completeness: Completeness of the requirements is the second important characteristic of the software acquisition requirement. [13] defines the completeness of the requirement in two ways. According to the first definition software acquisition requirements document is complete "if everything that the software is supposed to do is included in the SRS or responses of the software to all realizable classes of input data in all realizable classes of simulations is included". In reality this definition can not be used to measure completeness as there are infinite numbers of possible requirements of a system. Second definition says that a software acquisition requirements document is complete if "responses of the software to all realizable classes of input data in all realizable classes of situations are included". The second definition does not provide a good solution for some less bounded problem domain. We use Alexander's requirements completeness model [1] to measure the completeness of software acquisition requirements document. There are four blocks defined in Fig 5.1. Thus all the requirements are divided into four sub-domains A, B, C, D. In the Fig 5.1 'We know' means the applicability of the requirement the problem domain. Thus 'We know a requirement' means that we know the requirement is applicable to our problem. In the same way 'We don't know a requirement' means that we don't know whether the requirement is applicable to our problem. Similarly 'We understand a requirement' means the meaning of the requirement is understood. In the Fig 5.1 block A [1] represents all the requirements that we know and we understand that the requirements are applicable to the problem. Similarly block C represents all the requirements that we know and we don't understand. Block B represents all the requirements that we don't know and we understand. In the same way block D represents all the requirements that we don't know and we don't understand. The process moves the requirements in block C to block A after clarification. Similar situation happens in case of block B and block D. According to [13] the completeness of the software acquisition requirements document is defined as Eq 5.2 [13] where n_A, n_B, n_C, n_D represent the numbers of requirements in blocks A, B, C and D of Fig 5.1 respectively. It is very difficult to measure the number of requirements in block C and D of the Fig 5.1. Thus Eq 5.2 shows a simplified version with requirements only from block A and B. Mathematically, $0 \geq Q_{completeness} \geq 1$ where a value 0 of completeness represents that all the acquisition requirements are totally incomplete a value 1 of unambiguity shows that all the acquisition requirements are totally complete.

$$Q_{completeness} = \frac{n_A}{n_A + n_B + n_C + n_D} = \frac{n_A}{n_A + n_B} \qquad (5.2)$$

3. Correctness: Lastly we measure the correctness characteristic of the software acquisition process. According to the [13] a software acquisition requirements document is correct "if and only if every requirement represents something required by the system to be built every requirement in the SRS contributes to the satisfaction of some need". Quality of the software acquisition requirement is measured by counting the total percentage of the acquisition requirements that have been validated. Thus Eq 5.3 shows how correctness of the requirements are calculated where n_C represents total number of correct requirements, n_{NV} represents total number of not validated requirements. For example an old requirement is a non validated requirement. More specifically $n_C + n_{NV}$ is nothing but n_r the total requirements of software acquisition.

$$Q_{correctness} = \frac{n_C}{n_C + n_{NV}} = \frac{n_C}{n_r} \qquad (5.3)$$

To summarize we say the three quality of the software acquisition requirements unambiguity, completeness, correctness are measured by Eq 5.1, Eq 5.2, Eq 5.3.

5.3.3   Technical Requirements

This process maps user requirements to technical definitions. We use Software Quality Function Deployment or QFD method [4] [17] to adapt front-end user requirements to backend technical requirements. There are five basic steps followed in the QFD model [17] shown in Fig 5.2.
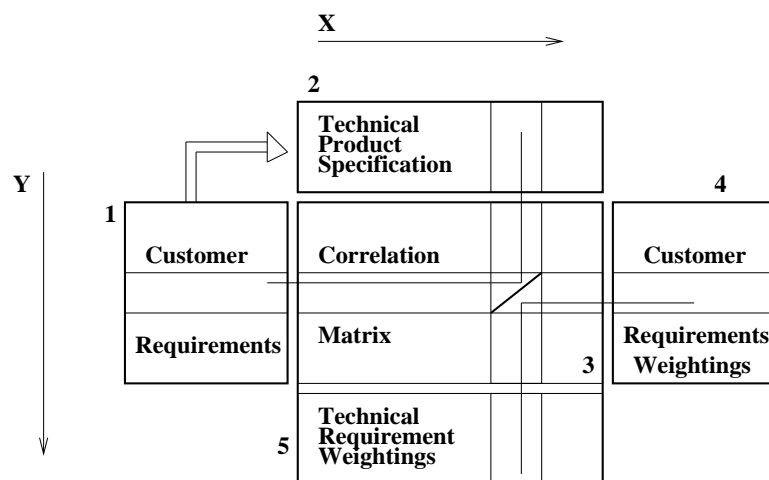
Figure 5.2: Steps in Quality Function Deployment (Borrowed From [17])

Step 1: Left y axis of the table specifies the user requirements. For example 'database' is a user requirements which is written in user's language.

Step 2: It transforms all the user requirements to technical requirements. Top x-axis stores the technical requirements. All the technical requirements are written in technical terms. For example an user requirement 'database' in the step 1 can be transformed to 'compatible to solaris' as a technical requirement. Some user requirements are transformed into multiple technical requirements. In case of different view points either two separate technical requirements are defined or two separate test cases are defined.

Step 3: Users build the correlation matrix between technical and user requirement. The correlation matrix stores the relationship between user and technical requirements. In case of many users, the decision of the majority is taken to fill up the relations in correlation matrix.

Step 4: A weight is assigned to each user requirement. The right y-axis stores the weights. Past experience guides weight selection. Afterwards, the IT group normalizes the weights.

Step 5: The technical requirement weights are calculated by summing the product of user requirement weights and the correlation matrix values between the user and technical specifications. Normalization of the technical requirement weights is also performed on a scale of 0-1.
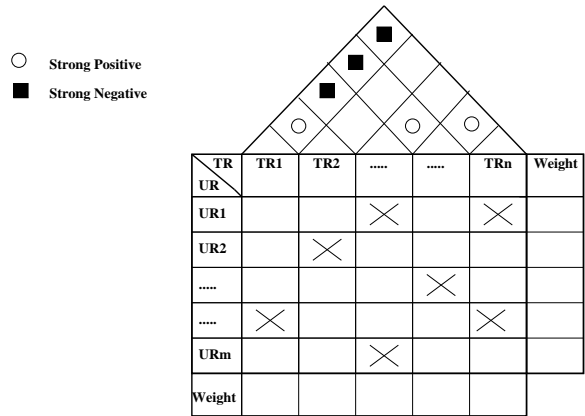
Figure 5.3: QFD Correlation Matrix and Weights Table

Fig 5.3 shows an example of standard QFD table with n technical requirements and m user requirements and corresponding column for user requirement weights and row for technical requirement weights. Lastly the outcome of QFD model [17] is a mapping between user and technical requirements, with user and technical weights. The relations between two technical requirements are also shown in the figure with two categories strong positive and strong negative.

5.3.4 Test Plan

The goal of the Test Plan is to provide the Analysis stage with an implementable suite of tests that yield a maximal coverage of user requirements. We say maximal, as constrains may prohibit an exhaustive plan. Factors such as time, cost, and technical feasibility will all play in to the coverage of the Test Plan. We expect that both IT staff and end-users will perform portions of the testing.

The Test Plan should specify:

1. **What is being tested**: specific user or technical requirements.

2. **Who will do the testing**: This may be a group, such as "IT group" or "Finance department users", or an individual with a particular set of requirements.

3. **How testers will perform the test**: The optimal test method, we believe, would be with use cases, since then tests are performed *in situ*. Tests may be performed out of context, but this may lead to a low test quality.

4. **Criteria for grading**: The Test Plan should be clear on how a tester should grade a product. In some cases, for instance, it may be important that a task be completed quickly by a user. In other cases, perhaps it is more important that input data be correct on first entry.

In general, we expect that tests for user functional requirements may either be tested directly, or derive from use cases. When testing directly, one should be aware that users from different groups may have different priorities that may affect how they grade the software. One should try an determine the existence of such divergent views and account for them in the Test Plan.

Before we move on to a few theoretical topics, we wish to call attention to the field of product testing [39]. Product testing is a well established field, made popular by such publications as *Consumer Reports*. As we mentioned above in the elicitation section, COTS/MOTS evaluation is essentially a black-box environment. One notes that the COTS/MOTS acquisition process, by definition, is closer to buying consumer goods than to specifying and developing a custom software package.

Based on Scriven [39] and Rivard *et al.* [35], we would recommend the following in devising a Test Plan:

1. Designate "Must Have" requirements, which lead to "Must Pass" tests. If a product fails these tests, there is no need to test further.

2. The "weighted sum" approach we use is problematic. If one uses a response scale of 1-5, no one requirement holds much weight when there are, for instance, 100 requirements. Alternatives would be to use a non-linear scale. For "Must Have" requirements, one could scale each to the sum of all other non-Must Pass requirements. Suppose, for example, there are 6 requirements: 2 "Must Have", one "5" and three "2" requirements. Then each "Must Have" would be worth "11".

3. As testing progresses, there will be a time evolution of how testers grade products. Scriven suggests that evaluators revisit early tests that show skew.

4. When considering a feature between multiple products, Scriven suggests that the most reliable method is head-to-head comparisons. While we have not specifically accommodated such tests, one could use head-to-head comparisons as an additional method to test for the same trait in a single-trait/multi-method test design. One could also interleave product tests rather than do one product at a time. Interleaving may have a higher cost, since multiple product environments must coexist.

5.3.5   Formal Experiment Design

There are many significant technical issues with test design and data analysis. In the interests of practicality, we have avoided complicating the testing process with rigorous experimental design, but we do wish to mention several points. The main point is that user functional testing is primarily a social science endeavor. Human-computer interaction, perceived usefulness, perceived ease-of-use, and perceived functionality all include a substantial psychological and sociological factor. As our Test Plan is not an *Experimental* Test Plan, the statistical strength of our comparisons would be of dubious value in a formal setting. If one were to make an Experimental Test Plan, one would need to identify a *control group* of end-users and stake-holders and track them through the acquisition life cycle. One would also need to formulate a *null hypothesis*, such as "there is no difference in user satisfaction between current practices and practices automated by COTS/MOTS software."[1] One would probably need several hypotheses. From such a design, one would have orthogonal means.

---

[1]User satisfaction is not a necessary and sufficient condition for a successful product. Rivard [35] identifies three spheres of quality.

One could also control the propagation of error. Confidence interval analysis could determine proper test-group size. For small groups, one could use methods like Power Analysis. The topic of factorial experiment design is covered at length in [27] from a social sciences view and in [25] from a computer science view.

Another theoretical issue is how grading the tests will translate in to a cumulative average grade for a product. Each test will encompass one or more technical requirements from possibly one or more user requirements. The user requirements are weighted on a response scale while technical requirements are a Bayesian partition of a user requirement. Traditional multitrait/multimethod analysis proceeds along the lines of [9]. Given a set of tests that covers, with possible overlap, a set of traits, one may deduce the significance of the tests and discern validity. Test scores may be averaged and then applied via their partition weights to the user weighting. One may also use internal consistency checks, such as Cronbach's alpha [12, 26]. There are also various forms of Analysis of Variance (ANOVA), Fisher's test, and others, which provide similar significance tests.

If one were to have users assign a Likert scale to requirements, one could perform tests on the requirements to determine the inter-correlation of requirements to product description. One could then find high t-test requirements, which could indicate ambiguous requirements.

The stochastic partitioning of a user requirement in to one or more technical requirements imposes certain constrains on the system. First, probability not assigned to the presence of a trait is assigned to its negation. Second, the absolute probability is dependent on the frame of discernment. If one views a trait among only a few other traits, it will have a higher absolute probability than if it is placed among many other traits. We note purely as a matter for further research that a Dempster-Shafer [5] model might be more appropriate for user functional requirements. In such a model, we no longer need to normalize (sum to one). Also, one may assign uncertainty to overlapping sets of events.

Finally, we note that our Test Plan tacitly assumes Independence of Irrelevant Alternatives (IIA) and that individual preferences coincide with group aggregate choice making [14]. We assume that if individual testers grade a product in a certain way, then the user groups in aggregate would make likewise choices. This is not necessarily true. When comparing more than two products, one does not necessarily have pair-wise independence. If we test products X and Y and then introduce product Z, we are not guaranteed that Z will decrease the values of X and Y in proportion to their ratio. Since we have no assurance of tester homogeneity in grading frequency, it is unlikely that individual choices will aggregate to the group level. In summary, we note that a group may choose differently than the average of individuals and that correlation between alternatives may skew results. If we grade products X, Y, and Z, then remove Z, the grades for X and Y may not be correct given that choice set.

5.4   Use of Database

In the Requirements section, we believe the main benefit of a database would be to track user requirements and how they were translated into test cases. In the Analysis section, one may then record the variance of test cases. For requirements reuse, one would then have a database of requirements, their test cases, and an idea of requirement quality.

As user requirements change and become more refined with time, one should also record this progression in a database. The main purpose, again, is to facilitate requirements reuse.

If one is starting a new project, one may lookup early requirements in other projects. This may give IT personnel an idea of where their early requirements may lead.

5.5   Exit Criteria

Since the Choice Phase is iterative, it is not necessary to have an absolute completion of the Requirements stage. Below we list our recommendations on determining if the process is ready to move to the Analysis stage. With each iteration of the model, the model user should interpret these criteria more narrowly.

1. **User Requirements**: On the first pass, the end-users should not be able to identify any more "5" requirements or any more "Must Have" requirements. On successive iterations, the number of new requirements should decrease. Because there may be some capabilities-to-requirements interaction, the number of new requirements may not ever reach zero, but the net gain/loss of requirements should approach zero.

2. **Quality**: The quality analysis should be completed and be of an acceptable level for the current model iteration. We do not have any guidelines, since these would be based on experience with the model.

3. **Technical Requirements**: The QFD process should be complete and there should exist a mapping of all existing user requirements to testable technical requirements.

4. **Test Plan**: All "Must Have" requirements must have one or more test cases. In the early iterations, perhaps only the higher priority items need test cases. In later iterations, there should be an increasing coverage of test cases.

6. Formal Reviews

6.1   Purpose

Formal reviews are evaluations of project status to ascertain discrepancies from planned results and to recommend improvement. Such activities help find and solve problems as early as possible. In our model, we conduct formal reviews after requirement, analysis, customization, and pilot stages.

As defined in [21], a review process can be divided into two parts: management review and technical review. Both of them can happen during the examined stage, or after that stage.

The objective of a management review is to provide recommendations for the following:

1. Making sure that activities progress according to plan, based on an evaluation of project status. For example, if an evaluation of the requirement stage gives a satisfactory result, then the next step, analysis on products can start.

2. Changing project direction or identifying the need for alternative planning. For example, review on analysis may determine that no current solution meets specific user requirements. After discussion with the customer, the review board may require the IT group to modify the requirements.

The objective of a technical review is to verify the correctness and standardization of conducted activities. If the review board found errors in the output or the examined process, the IT group might be required to repeat the process. Output of this stage is a review report identifying the following [21]:

1. The project being reviewed;

2. The review team;

3. Inputs to the review;

4. Review objectives;

5. Action item ownership and status;

6. A list of issues and recommendations identified by the review team that must be addressed for the project to meet its milestone;

7. Recommendations regarding any further reviews and audits, and a list of additional information and data that must be obtained before they can be executed.

6.2   People

There would need to be a formal review board formed by:

1. **IT Group**: for technical review

2. **Management**: for management review

3. **Stakeholders**: for both.

6.3   Procedures

The following recommended steps are adapted from the IEEE standard on formal review [21].

6.3.1   Overview

A qualified person from the project under examination shall conduct an overview session for the review team when requested by the review leader.

6.3.2   Preparation

Each person on the review team individually studies the material and prepares required presentations for the review meeting.

6.3.3   Examination

During the management review the review team holds one or more meetings to:

1. Examine project status and determine if it complies with the expected status according to a predefined plan;
2. Examine project status and determine if it is overly constrained by external and internal factors not originally considered in the project plan;
3. Record all deviations from the expected status accenting risks;
4. Generate a list of issues and recommendations to be addressed by higher level management;
5. Generate a list of issues and recommendations to be addressed by other responsible individuals, or organizations who affect the project;
6. Recommend what course of action should be taken from this point on;
7. Recommend authorization for additional reviews or audits;
8. Identify other issues that must be addressed.

During the technical review, the review team needs to:

1. Examine the element under review and verify that it complies with the specifications and standards to which it must adhere. The team needs to record all deviations from the specifications and standards.
2. Document technical issues, related recommendations, and the individual responsible for getting the issues resolved.
3. Identify other issues that must be adhered.

6.3.4   Rework

If the review board decided that the process or the output of the stage under examination did not meet the requirement, it will require the execution group of reviewed stage to rework on that stage.

6.4   Use of Database

During the review process, the review board will present to the execution group the problems found. The acquisition process would be able to go on only after the problems are solved. The problems and the solutions should both be recorded into the database. Later reviews could benefit from the history data by first focusing on the most common problems previously encountered, and by providing suggestions to the execution group based on the previous solutions.

6.5   Exit Criteria

The management review is considered complete when:
1. All issues identified in the review statement of objectives have been addressed;
2. The review report has been issued.

6.6   Elements for Review at Specific Stages

For requirement:
1. Output documents specified in the requirement chapter;
2. Process of requirement elicitation, bidding, and calculating the estimation.
3. SRS quality and Technical Requirements formalization.
 For analysis:
1. Output documents specified in the analysis chapter;
2. Product grade summary table.
 For customization:
1. Output documents specified in the customization chapter;
2. Process of development, testing and deviation analysis, and inspection.
 For pilot:
1. Output document specified in the pilot chapter;
2. The type and severity of problems in the pilot program. If they cross a customer defined threshold, then the process should implement mitigation plan.

7. Analysis

7.1   Purpose

The Analysis stage achieves three goals: estimating an equivalent custom development effort, acquiring vendor proposals, and testing vendor products for conformance with user requirements.

This section depends on the Weighted User Requirements, Use Cases, Test Plan, and Weighted Technical Requirements documents produced in the Requirements stage.

The Analysis stage will produce the following documents:

1. **Qualified Vendor List**: a list of potential vendors.
2. **RFP**: the Request for Proposal document sent to qualified vendors.
3. **Proposals**: vendor proposals in response to the RFP.
4. **Test Results**: the responses of graders to the test cases specified in the Test Plan. Includes some analysis of response variance as a detection mechanism for requirement ambiguity, poor test cases, or undetected multiple viewpoints.

7.2   People

1. **IT Group**: All sections.
2. **User Groups**: As testers.
3. **Customer**: Possibly involved with vendor RFP process.

7.3   Procedures

The analysis stage is made up of three distinct activities. It is possible that they may be conducted in parallel. The first activity is to conduct a Custom Development cost estimate. This estimate is compared against vendor costs. In later iterations of the model, it might be used as a basis for commencing a custom development effort if the acquisition endeavor does not find suitable COTS/MOTS software. The Vendor section creates an RFP to send to vendors and also suggests the structure of vendor responses. The In-House Testing section performs user requirement testing against demonstration versions of vendor software.

7.3.1   Custom Development Estimate

With each iteration of the Choice Phase, the IT group should produce more refined cost and schedule estimates for a full custom development project. We would recommend a method such as Function Point Analysis [42], which does not require a detailed knowledge of the software project. Nelson *et al.* [32] have some useful high-level methods when considering how to acquire an application. We consider this part of the model a contingency plan in case the IT group does not find any suitable COTS/MOTS software.

On the first iteration, the custom development estimate is mostly a "sanity check" to make sure that packaged software is a cost savings. At this point, we would not expect custom development to be a viable alternative. After several rounds of iteration, custom development might be a necessary course of action. If one has iterated the model so many times and not found a pre-packaged solution, it is possible that nothing exists to

meet the particular customer needs driving the process. By this point, the estimate for custom development should be rather refined. The IT group may also contact an outside development company and ask for bids on the project.

The custom development estimate should include:

1. **Estimated labor costs**: If using internal resources, the labor costs should include a pro-rated salary and facilities overhead.

2. **Estimated delivery schedule**: If using internal resources, this should consider regular work duties of personnel or add cost to relieve development staff of regular duties.

3. **Estimated risk**: Based on the IT staff's familiarity with the application domain and expected development environments, one should rank both the Cost and Schedule estimates as High, Moderate, or Low risk. If the acquirer were to use an outside development group, the acquirer should appropriately adjust the risk. Here, risk is the chance of significant over-cost or over-schedule. The customer should define "significant."

7.3.2   Vendor Involvement

Vendors will play an important role in the acquisition process. While our model does not address how to identify potential vendors, we imagine that it will be a combination of trade journals, trade shows, news groups, individual knowledge, and a bit of detective work. A vendor may be a first-party manufacturer, a Value-Added Reseller, or a skilled systems integration or consulting firm. This section describes the interplay between the acquirer and the vendors. Two IEEE documents [22, 23] also define the acquirer/vendor relationship and prescribe actions. We take a less formal approach.

After the IT group has a list of potential vendors, it will begin an informal process of contacting the vendors to assess their potential. Although we say "informal", the IT group should maintain written records of all vendor contact. The IT group should prepare a high-level problem description, perhaps similar to the original customer description from the Preparation stage, or use a subset of user requirements. The IT group should present the high level description to the vendors and have the vendors make an initial suitability estimate. We will call the group of vendors who show sufficient interest and whom the IT group believe suitable the "Qualified Vendors."

We doubt the next steps will ever follow the same path twice, so we will simply outline important points. The acquirer should prepare a detailed Request for Proposal (RFP). The RFP should include all "Must Have" requirements and as many use cases as reasonable. If the use cases were organized hierarchically, then the first round RFP could only include top-level cases. The qualified vendors should then be given a reasonable amount of time to respond. It may take a month or more for complicated RFPs. There may be significant informal exchanges between the qualified vendors and the acquirer [10]. These exchanges would refine the vendors understanding of the problem and focus their responses.

The RFP process will also help the IT group refine the requirements. The process of writing the RFP should bring to light poorly understood requirements. It may also reveal requirement associations. Vendor queries about RFP items should also feedback to the requirements. If a vendor does not understand the meaning or import of a requirement, it is ambiguous and should be refined. If a vendor response to a requirement is off target, the requirement is ambiguous and should be revisited. One would, of course, contact the

vendor about these issues since it could have been more a vendor misunderstanding than a poor requirement [1].

   Qualified vendor responses to the RFP should provide:

1. **Executive Summary**: A brief summary stating if the vendor's product is suitable. If the software is suitable, the summary should include a total one-time cost, any recurring costs, and a delivery schedule. If there are any modifications or customizations required, the summary should state such. The vendor should also state any assumptions or conditions of delivery and implementation. The vendor should elaborate on any such assumptions or conditions elsewhere in the response.

2. **Detailed Costs**: The vendor should state their charges. These will include initial software cost, the cost of per-user licenses considering the estimated deployment size, and maintenance plans. If there are any needed product customizations, the vendor should estimate a reasonable implementation cost if the acquirer were to out-source the labor. If there are any product modifications necessary, the vendor should state their labor costs and pricing structure (fixed cost, time & materials, etc.).

3. **Detailed Schedule**: The vendor should provide an implementation schedule. If there are customizations required, then the vendor should estimate a reasonable labor effort. If there are modifications necessary, then the vendor should state their implementation duration.

4. **Training Plan**: The vendor should provide recommendations on the amount and kind of training end users and IT staff will need. This section may include possible providers. Costs should be in the Detailed Cost section.

5. **Detailed responses to RFP questions**: For each use case or requirement presented in the RFP, the vendor should state if and how their product meets said requirement. If customization and/or modification is needed, the detailed response section should so indicate. The vendor response should also give time and cost estimates for such changes. These may be multiply referenced under the Cost/Schedule sections and the Detailed Response section.

7.3.3   Demonstration Software

   We consider it almost essential that the acquirer obtain demonstration versions of software from vendors. We avoid saying that it is required, as it may be possible to make a decision without seeing or testing a package. Some packages may be so large or complex as to prohibit demonstration software. In these cases, the acquirer should make a field trip to the vendor's site to see demonstrations there.

   Our system is based largely on end-user evaluation of potential COTS software. In cases where demonstration software is not available, the acquirer and vendor should try to arrange for end-user exposure to the software such that they may make some form of evaluation.

   When demonstration software is available, the acquirer and vendor should work closely to install and configure the demonstration system. In the interests of time, it may not be practical for the IT group to gain a sufficient understanding of all products to construct

---

[1]Obviously, there was some problem with the requirement since it was open to vendor misunderstanding. In the interests of practicality, we would not spend too much time trying to "fix" a requirement that was largely correct

fair test environments. It is important to the evaluation that each product be tested in a properly installed environment.

7.3.4   In-house Testing

In house-testing lists and analyses the test results obtained from various test suites. Occurrence of any discrepancy in the results are verified by calculation of the variance. Following steps are followed in in-house testing.

1. **Product Grade Table:** The produce grade table stores the grade results obtained from various acquisition tests. Fig 7.1 shows the components of the product grade table. It stores the individual grade results of each tester for every product of every test suites. The table is sub-divided into several product blocks (column wise) to store the results for different products X,Y,...,Z. Similarly a single column in a particular product block stores the grade results obtained from a particular tester for all the test suites. For example in Fig 7.1, a particular block is assigned for a particular product X. The product block X stores the results obtained from n testers: 1,2,...,n in its different columns whereas a particular row in a product block stores the grades results from all the testers obtained for a particular test suite. A single row stores the test results for all the M test suites A,B,...,M. Fig 7.1 also shows how the table stores a grade 'B' obtained from tester 2 for test suite A of product Y. It is to be noted that not all testers grade each test. Tests are graded in terms of five grades A, B, C, D, F where on a scale A corresponds to 4.0, A- to 3.7, B+ to 3.3, B to 3.0, B- to 2.7, C+ to 2.3, C to 2.0, C- to 1.7, D+ to 1.3, D to 1.0, and lastly F to 0.

Product / Test \ Tester	X					Y					..					..					Z				
	1	2	..	..	n	1	2	..	..	n	1	2	..	..	n	1	2	..	..	n	1	2	..	..	n
A							B																		
B																									
..																									
..																									
M																									

Figure 7.1: Product Grade Table

2. **Product Grade Summary Table:** Product grade summary table is a concise from of the product grade table. Fig 7.2 summarizes the components of the product grade summary table. Product grade summary table stores the mean and variance of the grade results obtained from all the testers for every product and also for every test suites. The table is sub-divided into several product blocks (column wise) to store results for different products X,Y,...,Z. Similarly each product block consists of two columns: mean and variance. Mean and variance are calculated from test results of all the testers. More specifically these are based on a particular test suite and also for a particular product. For example in Fig 7.2 a particular block is assigned for product X. In the product block X, mean and variance columns store the mean and variance of the grade results obtained from n testers 1,2,...,n. Similarly a particular row in a product block stores the mean and variances the product for a particular test suite. Test results for all the M test suites A,B,...,M are stored in each row.

Product Test / Statistics	X		Y		..		..		Z	
	Mean	Var	Mean	Var	Mean	Var	Mean	Var	Mean	Var
A										
B										
..										
..										
M										

Figure 7.2: Product Grade Summary Table

3. **Variance Checking:** A upper cutoff point of the variance are used to check any discrepancy of the results. If the variance crosses the cutoff point the results are analyzed, requirements and test suites are reviewed. A high value of variance occurs in case of an ambiguous or an incomplete or an incorrect requirement. Thus in case of large discrepancy in the results, the IT group must decide if they will go back and clarify requirements or change the test suite. Certain social influences may give rise to multiple viewpoints on essentially the same software feature. If these viewpoints were identified in the elicitation stage, then the appropriate end-user should test for a particular viewpoint. If use differences were not detected, then we expect to see a high variance in grades for a particular test. This high variance would be an indicator that there is either an ambiguous requirement, an ambiguous test, or there exist undetected multiple viewpoints. One would need to determine the cause through tester debriefing. Lastly we say on a 0-4 scale grade scale variance≥ 0.5 is probably significant.

7.4   Use of Database

The Analysis section should record test case results to associated test case entries made in the Requirements section. This tracks the performance of user requirements to test cases to actual grading.

The Analysis section should also maintain a vendor database. By maintaining a listing of vendors and products evaluated, the acquirer may, at a future time, find potential products for new acquisition projects.

7.5   Exit Criteria

To exit the Analysis stage, the IT group should have:
1. Sent out RFP and received responses, as appropriate. These steps are not required in every iteration of the model.
2. Performed in-house testing as needed by the current model iteration. The IT group should have a complete Product Grade Summary Table.
3. After each iteration, the IT group should have an improved cost and time estimate for custom development. At IT group discretion, an outside development firm could be contacted to bid on the development.

8. Presentation

8.1 Purpose

At the presentation stage, the main purpose is to communicate with the customers-to help them understand the results of analysis, and to solicit their opinions while making the decision on choosing the most suitable product.

At the analysis stage, the IT group evaluates each candidate product based on the test plan produced at the requirement stage. At the presentation stage, the group will transform these results into overall scores reflecting to what extent the products meet the requirements. These scores, combined with the cost estimation and development schedule for each product produced at analysis stage, will help the customers decide which product to choose.

Output of the presentation stage include:

1. **The decision on purchase or iteration.**

2. **Documents explaining the decision.**

8.2 People

1. **IT Group**: All sections.

2. **Customer**: All sections.

3. **Management**: Purchase decision.

8.3 Procedures

The IT staff will need to generate the overall evaluation on each candidate product based on the analysis results. The evaluation data and further data such as cost and schedule estimation may be presented using Multiple Metric Graphs [33]. At the end of this stage, customers will help to decide whether to purchase a product or iterate the previous stages.

8.3.1 Generate Overall Evaluation Based on Analysis Results

From the analysis stage we got the test result of each product based on each test case. The result can be shown in a table such as Table 8.1.

	Product X			Product Y			...
	tester1	tester2	tester3	tester1	tester2	tester3	
testcase1	A	B+	B	B-	B	A-	...
testcase2	A-	B	C	B+	B+	A	...
testcase3	B+	B	B	C	A	B+	...
testcase4	A-	A-	B+	C+	B+	A	...
...	...	...	...	...	...	...	...

Table 8.1: Product Grade Table

	Product X	Product Y	...
testcase1	3.43	3.13	...
testcase2	2.90	3.53	...
testcase3	3.10	3.10	...
testcase4	3.57	3.20	...
...	...	...	...

Table 8.2: Product Grade Summary Table

	Product X				Product Y				...
	TC1 (0.6)	TC2 (0.4)	TC3 (0.5)	TC4 (0.5)	TC1 (0.6)	TC2 (0.4)	TC3 (0.5)	TC4 (0.5)	
TR1	3.43	2.90			3.13	3.53			...
TR2			3.10	3.57			3.10	3.20	...
...	...	...	...	...	...	...	...	...	...

Table 8.3: Relationship between Technical Requirements and Test Cases

	Product X	Product Y
TR1	3.22	3.29
TR2	3.34	3.15
...	...	...

Table 8.4: Score for Each Technical Requirement on Each Product

Several testers were assigned (three for the example above) for each test case on each product. We can translate the grades to numbers as introduced in the analysis stage. A mean value can be produced as an overall score for each test case on each product, as shown in Table table:mean.

Since each test case is related to a specific technical requirement-each technical requirement can be tested using one or several test cases, we can get the evaluation for each technical requirement on each product. Let's assume that technical requirement 1 (TR1) generated two test cases: TC1 and TC2, and technical requirement 2 (TR2) generated two other test cases: TC3 and TC4. Based on their relevance to the technical requirements, each test case was assigned a weight. We can thus calculate a value for each technical requirement on each product in the manner shown by table 8.3 and 8.4.The weight of each test case is also shown after the test case names:

$V_{TR1} = V_{TC1} * 0.6 + V_{TC2} * 0.4$
$V_{TR2} = V_{TC3} * 0.5 + V_{TC4} * 0.5$

(V_X means the evaluation value of X, the numerical figures in the equations represent the weights of the test cases)

Since what is meaningful to the customers is the user requirements instead of technical requirements, we still need to calculate the evaluation score for each user requirement (UR). We can do this in a similar manner. Like the relationship between technical requirements and test cases, each user requirement generated several technical So that we have the

	Product X				Product Y				...
	TR1 (0.3)	TR2 (0.3)	TR3 (0.4)	TR4 (1.0)	TR1 (0.3)	TR2 (0.3)	TR3 (0.4)	TR4 (1.0)	
UR1	3.22	3.29	3.30		3.34	3.15	3.20		...
UR2				3.70				3.50	...
...	...	...	...	...	...	...	...	...	...

Table 8.5: Relationship between User Requirements and Technical Requirements

	Product X	Product Y
UR1	3.27	3.23
UR2	3.70	3.50
...	...	...

Table 8.6: Score for Each User Requirement on Each Product

following tables and calculation (each technical requirement is also assigned a weight). Note that TR1, TR2, TR3 corresponds to UR1, while TR4 corresponds to UR2:

$V_{UR1} = V_{TR1} * 0.3 + V_{TR2} * 0.3 + V_{TR3} * 0.4$

$V_{UR2} = V_{TR4} * 1.0$

This information can be presented directly to the customers. Since each UR was also assigned a weight by the customers at the requirement stage, we can generate an overall score for each product. For the example shown in the table above, if we assume that there are only two user requirements UR1 and UR2, and they are given the weights of 0.7 and 0.3 respectively, we can get the overall score of 3.34 for product X, and 3.31 for product Y.

A product with a higher overall score generally meets the customers' needs better, but it is not always the case, especially when the scores of different products are quite close. In such situations, customers might need to look at the score for each requirement instead of the overall one.

8.3.2 Deviation Analysis

We can see from the tables introduced in the previous section that sometimes there are big differences between the scores from different testers on the same product. This might come from the different preference of the testers. For example, a division wants to buy game software. Overall,the game should be "cool", "not boring". There might be two types of candidate products, one is A: strategy games, the other is B: action games. Testers preferring strategy games would tend to give higher scores to A, while those preferring action games tend to give higher scores to B.

In the previous section, when we were calculating the overall score for the different products, we ignored the deviation of the real scores from the mean values. But if we get a mean value of 3.0 for A product, and 2.5 for B product, although it seems that A is better than B, when we consider the deviations, the judgment might not be true. If A has a deviation of 1.0, and B has a deviation of 0.5, then there is no statistical difference between the two.

To be accurate, starting from Table 8.1, we can calculate the standard deviation for the mean score of each test case on each product, and carry it through the following calculation. Let variance=$\frac{1}{n}\sum_{i=1}^{n}(X_i - \mu)^2$, where $X_1,...,X_n$ represent the grades from different testers, and μ represents the mean value. Then the deviation $\sigma = \sqrt{var}$.

When calculating Table 8.4 from Table 8.3 or Table 8.6 from Table 8.5, we can apply the following formula to get the deviation values:

$$\sigma_{new} = \sqrt{\sum_{i=1}^{n}(W_i * \sigma_i)^2}$$

W_i here means the assigned weight for the specific technical requirement or user requirement.

When the overall evaluation of a product presents a high deviation, then that means quite different opinions exist for the performance of the product. The IT group need to investigate the reason behind it, and solve the problem.

8.3.3   Score, Cost and Schedule Presentation

Besides the overall score, estimation of the cost and the schedule are also important measurements. These are the main metrics for evaluating and comparing the candidate products.

A good way to look at the metrics is using a multiple metric graph, which was introduced in [33].

8.3.4   Multiple Metric Graphs

A Multiple Metric Graph displays characteristics on slices of a large, circular pie. The pie is divided into unequal slices, one for each measurement or characteristics to be presented, and the size of the slice (e.g., the number of degrees in the arc) represents the importance of the metric. Thus, the larger the slice, the more important the metric.

The inner circle represents a goal or minimum, and the outer circle represents a maximum. Within each slice, a point is placed in the center (equidistant from the adjacent radii) to represent the degree to which the goal is met. The center of the pie represents the best case; the outer edge of the pie is the worst case. In the situation of presentation, suppose we use A=Score, B=Cost, and C=Duration of schedule as the metrics.

We also assign a weight for each of them. The metrics must be chosen so that the smaller the number, the better. The customer may determine that B is most important, A second, and C third. When asked to weight the three metrics, we can assign 50 points to B, 30 to A and 20 to C. Correspondingly, we draw a multiple metrics graph with slices of 180 degrees (0.5*360degrees), 108 degrees, and 72 degrees, respectively, as shown in the figure below:

We place a point in each area to represent the matching of the product to the corresponding requirement. Next, lines are drawn from each of the representative points to the places where the goal arc meets the edges of the slice. In this way, the area of the resulting quadrilateral for each metric represents the degree to which goals are met: the smaller the area, the better. The polygon formed by uniting the three quadrilaterals represents the overall quality of the products. If there are too many requirements to be represented by the slices of the pie, requirements can be categorized into different groups, and the each group can be represented as a metric on the graph.
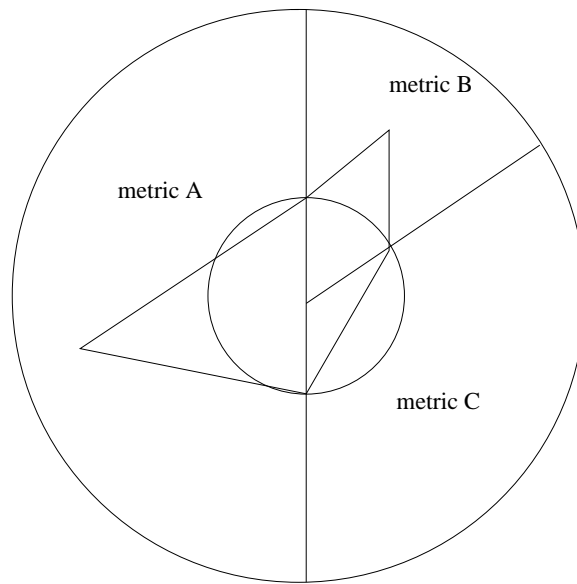
Figure 8.1: Multiple Metric Graph

To compare the quality of different products, we can draw a polygon for each product on the same map, then it becomes easier to evaluate the strengths of different products and make judgment based on customers' preference.
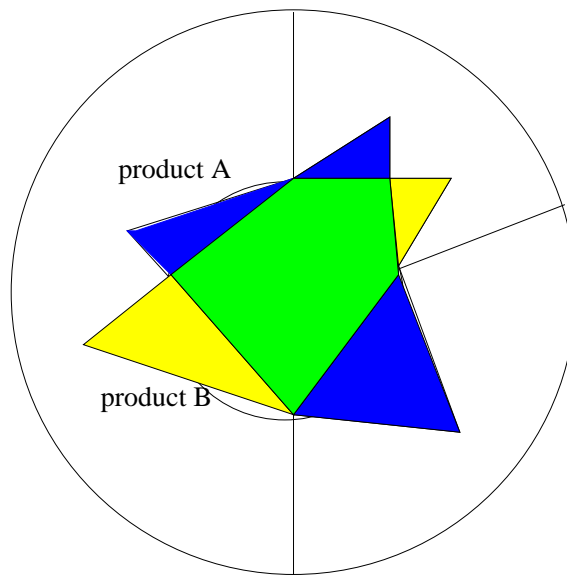
Figure 8.2: Comparing Metrics of Two Products

8.3.5   Present Further Information

Customers might already be able to make judgments based on the overall scores and the multiple metric graphs. It is also quite possible that they want to look at more detailed

information about the products. For example, they might want to review the process of calculating the overall score, or they might want to try the demos of the products to get a clearer idea on what the products looked like and how they function. IT staff should be prepared to present them the calculation tables and the demo of the products.

Besides evaluation score, cost estimation, and schedule estimation, the IT group also needs to make an assessment of the risk factors such as the size and the financial stability of the vendor companies. High risk of a vendor company means a high possibility of losing money for the acquirer. The risk assessment of vendors should also be presented to the customers to assist their selection.

8.4   Collect Feedback from Customers

After the customers have understood the analysis results and have viewed the demos of the products, they may know whether their requirements are met or not. If there are several comparable candidate products, customers can also help to choose one according to their special preference. IT staff should collect such information by meetings, emails, or other form of communication.

8.5   Make the Decision

After presenting enough information and collecting customers' feedback, it is time to make the decision. Besides IT staff and user representatives, management might also get involved to make the decision.

If the customers prefer one of the candidate products, then a decision can be made to start the purchase process. If the customers are not satisfied with any of the product, then we must go back to refine either the requirement or the analysis. If the customers prefer several candidates and can not make the final choice, then nonfunctional considerations (such as relationship with the vendor) may be taken to assist the selection.

8.6   Use of Database

While selecting the most suitable vendor, background data of the candidate vendors would be useful for risk assessment. Such information should has been maintained by the acquirer in the database.

The IT group should record the data on what problems they met and how they solved the problems. They can also refer to the previous records for the solution of current problems.

8.7   Exit Criteria

At the end of the presentation stage, an agreement between the customers and the IT Group should be formed. It is either a decision of which product to purchase, or a decision to iterate back to the previous stages due to the difficulty of selection, or they might decide to stop the acquisition due to the fact that none of the current products can satisfy specific important needs.

9. Purchase

9.1 Purpose

In the previous stage the acquirer has chosen the vendor of the software product. At this stage the acquirer and the vendor will negotiate and sign a purchase contract.

This is the point at which an acquirer's wants and needs have to be reconciled with the vendor' capabilities and desires. What's best is usually that all of the wants are met by a strong supplier who is capable of dedicating the resources needed to make the project a success.

Output of this stage include:
1. **Legal contract.**
2. **Off-the-shelf software product which might need further customization.**
3. **Supporting documents to the software**

9.2 People

1. **IT Group**: All sections.
2. **Legal representative**: Legal issues.
3. **Management**: Purchase contract.
4. **Vendor**: All sections.

9.3 Procedures

The acquirer and the vendor will negotiate on relevant issues, and then sign the purchase contract.

9.3.1 Negotiation

There are many approaches to negotiation. The approach introduced in this section is to break the negotiations into "technical" and "commercial" parts. The following paragraphs on negotiation are based largely on [29].

It should be noticed that, too many times, vendors accept conditions that cannot be met in order to secure an order. When this condition becomes apparent, the project is headed for trouble [7].

Technical Negotiation

The specifications should be negotiated first, because their resolution could change the price. If special materials of construction are needed, the price may increase. If an acquirer is willing to accept a standard offering, the price may go down.

Future upgrades and technical support should also be identified to decide the price. Shifting markets, mergers and buyouts, or unforeseen technological developments can convert a vendor's best intentions on the product's capabilities and support into broken promises. Any project that doesn't allow for such contingencies as product substitution or escrow of a failed vendor's product is on a course to disaster [7]. The need on escrow depends on the existence of alternatives, and the assessment of the risk factors.

Commercial Negotiations

All items that will appear in the purchase order must be discussed and agreed to, especially performance specifications, test conditions, and remedies. After negotiations are complete, there should be no surprise for the acquirer or the vendor. Negotiations should be based upon the existence of adequate written specifications, a definition of the obligations and responsibilities of the supplier and acquirer, the time frames in which the work is to be accomplished; and a balance of the responsibilities, risks, and benefits to both parties.

During the negotiation process, the acquier and the vendor need to consider:

1. a means of avoiding disputes and of resolving disputes that arise;
2. investing only a minimum amount of funds before the quality of the vendor's work or product is demonstrated (at the analysis stage, the quality of work for different vendors have been initially investigated);
3. maximum total price, payment amount, or total value of the contract.

If negotiations with the selected vendor fail to produce a contract that will assure delivery of a quality product on time and properly supported, consider opening negotiations with an alternate vendor.

9.3.2   Purchase order/purchase contrast

There are two general recommendations. First, the acquirer should have his contract forms reviewed and modified by expert outside legal counsel who have experience in litigating contracts. Second, the acquirer should use his stationery and his contract as a starting point.

Based on an IEEE recommendation [22], the following steps can be followed in this process:

1. Determine the quality of the work;
2. Determine how payment is to be made;
3. Determine nonperformance remedies;
4. Prepare contract provisions;
5. Review contract provisions with legal counsel;

9.4   Use of Database

During the negotiation, history data on the cost and schedule estimation of the previous acquired products would be useful to refer to. Also, information on negotiated cost and schedule estimation for this acquisition should be recorded for further use.

9.5   Exit Criteria

At the end of the purchase stage, the purchase contract should have been signed. The software product (not customized) and supporting documents should have been prepared by the vendor.

10. Product Modification & Customization

10.1 Purpose

Since COTS software is originally designed for an overall marketplace, its functionality and performance may not completely fit the requirements from a specific user group. Before the production operation of the software, the acquirer may need to customize the software through APIs or scripting languages, or the vendor may need to modify the source code.

For the acquirer, the focus of this stage is the control on the schedule and on the software quality. Quality control is especially important for modification, since changing of source code is quite likely to incur new defects on the software.

After this stage, the software product will be ready for real use. The following documents will also be produced:

1. **User Manual** reflecting the current characteristics of the software product.
2. **Technical document** reflecting the current technical detail, which is useful for future maintenance.

10.2 People

1. **IT Group**: All sections.
2. **Vendor**: All sections.
3. **Customer**: Evaluating modified product.

10.3 Procedures

In this stage, the IT group should work together with the vendor's development group, performing the equivalent of a "receiving inspection" upon initial COTS receipt [7]. They would plan and monitor the customization or modification process, and then test the modified product based on the test plan produced at the requirements stage.

This practice ensures that the COTS product really does what it is expected to do. The process should adopt dynamic, risk-driven spiral-type models, assessing risks via prototyping,benchmarking, reference checking, and related techniques, and using top people to resolve top risk items in advance [7].

10.3.1 Plan the Process

Based on the difference between current product and user requirements, the IT group and the vendor's development group can plan out a list of changes, and then the development group can make a draft schedule. After discussion between the two groups, they can set up the milestones.

The groups should identify risks and make a plan based on risk assessment. For modification, one extra risk comes from the fact that defects are likely to be injected into the software. Analysis of defect distributions shows that the modules comprising software products often do not present uniform risk: some incur a disproportionate number of defects [28]. If these modules were known before entry to system test, software engineers could make more informed staffing, training, and scheduling decisions in preparation for system testing. A predictive model can be borrowed from [28].

In order to plan and execute the process effectively, a close relationship with the vendor is important. COTS vendor behavior varies widely with respect to support, cooperation, and predictability. An accurate assessment of a COTS vendor's ability and willingness to help with these areas is tremendously important. According to experience, mid-sized companies usually provide the best support [7].

10.3.2   Monitor the Process

The objective is to monitor the vendor's progress to ensure that all milestones are met and to approve work segments. Frequent meetings are necessary to check the milestones and current status. The IT group should consider the following when monitoring vendor's progress [21]:

1. The acquirer should provide all of its required deliverables (e.g., equipment, software, machine time, and reference materials) to the vendor within the specified time frame so that the vendor is not delayed.

2. Management should create an environment within the organization that supports the vendor's efforts. Internal disagreements should be resolved in-house by management and not left for the vendor to encounter.

3. The IT group should use measures of reliability and quality specified in the contract to evaluate the vendor's work.

4. The acquirer should provide some means for regular and continuous feedback to the vendor on vendor's performance in terms of overall progress and on handling problems. Informal reviews can be given more frequently, and formal inspection on progress status or milestones can be given less frequently. Undocumented informal communication with the vendor can lead to additional costs, due to the inconsistency of the real situation and the content of the contract. Any changes in the scope of work should be handled by amending the contract.

10.3.3   Testing

The IT group should test the delivered software product according to the test plan. If the delivered software does not meet expectations, then the IT group should submit the defects to the vendor. By applying the test plan on the modified product, the IT group can also evaluate and adjust the test plan, and use the refined test plan on further modified product, or on re-selection of the product. It is unlikely that as early as the requirement stage the IT group could anticipate the best way of acceptance testing for the vendor's product.

10.3.4   In-house Customization

For in-house customization, which is done by the acquirer, the process may be less formal than managing the vendor. The possible forms of conducting customization include: adding or removing functional components from the software using GUI tools; writing scripts using the high-level script language supported by the software. In such a situation, the IT group should develop an implementation plan and a test plan. They could use any accepted development methodology. Compared with developing a complete software, the activity of customization would have less uncertainty. Therefore plans could be made more accurate and practical.

10.4 Use of Database

After the customization or modification, the acquirer and vendor would know the real cost of the product and time spent on the project, which could be compared with the negotiated estimation. The acquirer's IT group should record this kind of analysis information into the database. Purchase negotiation and customization plan on future product can benefit from this information.

10.5 Exit Criteria

The exit criteria for customization or modification include passing the test and finishing updated user manual and technical document.

11. Pilot Deployment

11.1   Purpose

This stage of the software acquisition process verifies whether the software operates as needed in a real environment. This stage uses the test plan from the requirements stage. It produces a software evaluation report and a bug list.

11.2   People

1. **IT Group:** Involved in the definition and the analysis sub-stage.
2. **User:** Involved in the evaluation sub-stage.
3. **Vendor:** Involved in the analysis sub-stage to solve all the bugs reported.

11.3   Procedure

In this stage, the acquired software is deployed in a subset of the systems and tested by a subset of the all the user group members. The IT group performs their tests before general deployment to all users. This stage is subdivided in three sub-stages definition, evaluation, analysis.

1. **Definition:** In this sub-stage, the IT group states the technical and formal goals for pilot deployment.
   (a) **Pilot Group Size:** Pilot program group size is defined here.
   (b) **Pilot Group Member:** Pilot group members are chosen from users and IT group.
   (c) **Pilot Test Resources:** A selection of servers, printers, and other required items.
   (d) **Pilot Deployment Steps:** The steps to be followed by pilot group are defined.
   (e) **Pilot Test Suites:** Pilot test suites are defined in this stage. These should cover the main user requirements.
   (f) **Pilot Workload:** The definition stage verifies whether the pilot members have a suitable workload to test the user requirements.
2. **Evaluation:** In this sub-stage of the pilot deployment, the pilot deployment group from IT installs and configures the software. Pilot group evaluates the software on the basis of satisfaction of the requirements. It also grades the acquired software on the criteria of its easiness, efficiency and appropriateness for long term use. They also comment on the modifications needed. Risk associated with the software is also analyzed in the evaluation process. Underestimating the risks factors associated with COTS software often result in higher delay, higher maintenance cost. We use a risk mitigating model described in [45] which have faster review process and also lower maintenance cost.
3. **Analysis:** IT group analyze the results obtained from the pilot group. All the results are summarized according the requirements met or missed by the software. For each requirement missed by the software the degree is rated as a factor of minor, major, cosmetic.

(a) **Major:** A major miss must be fixed but can not be easily fixed.

(b) **Minor:** A minor miss must be fixed and may be fixed before or during deployment.

(c) **Cosmetic:** A cosmetic miss should be fixed but not a necessary change.

Thereafter the software gets a final evaluation. Analysis sub-stage is used to train the users, and to track all the problems through a central database for future use. The pilot group reports any bugs to the vendor.

11.4   Use of Database

The user evaluation is also tracked in the database. Similarly feedback obtained from pilot test suites are also stored. The database stores the name of product deployed with its version number, deployment time, deployment issues, the people involved in the deployment and lastly its dependencies. The number of copies deployed and their license limits are also stored in the database.

11.5   Exit

Exit from this stage takes place after the evaluation and the analysis sub-stages are complete.

12. General Deployment

12.1   Purpose

This stage of the software acquisition performs the deployment of the acquired software to users' systems. The general deployment stage produces progress reports for the acquisition team.

12.2   People

1. **IT Group:** Involved in installations and data reporting.
2. **User:** Involved in installations and data reporting.
3. **Vendor:** Involved in installations and data reporting.

12.3   Procedure

In this stage a deployment team installs the acquired software. They provide feedback to the acquisition process. This stage is subdivided into two sub-stages installations and data reporting.

1. **Installations:** The software is installed in the this sub-stage with the help of IT group, user and possibly the vendor. A centralized installation with the help of the software like LANDESK [19] or SMS [18] makes the deployment process more efficient as it reduces the person-time. The deployment team also facilitates any prerequisition, such as OS upgrades or new servers.
2. **Data Reporting:** The next important sub-stage is data reporting. Deployment information are collected from the deployment team and reported to the acquisition team. The acquisition team goes through the deployment data and verifies the deployment work. The information consists of the work performed, schedule information, cost expenses, and problem reports. The knowledge base system is also updated with the deployment information.

12.4   Use of Database

In this sub-stage, the database stores the name of product deployed with its version number, deployment time, deployment issues, the people involved in the deployment and lastly its dependencies. The number of copies deployed and their license limits are also stored in the database.

12.5   Exit

Exit from the this stage takes place only after the deployment of the acquired software along with the installations and upgrades of all necessary components are complete.

13. Maintenance

13.1 Purpose

The maintenance stage uses the user requirements from the acquisition process and previous periodic reviews from this stage. This stage maintains the software after the acquisition, for the duration of lifetime of the software in the user's group. The IT group produces periodic requirement review documents.

13.2 People

1. **IT Group:** Involved in routine maintenance, troubleshooting and end-user review.
2. **User:** Involved in end-user review.
3. **Vendor:** Involved in troubleshooting and end-user review.

13.3 Procedure

Maintenance is the last stage of software acquisition process model. This stage is categorized into three sub-categories routine maintenance, troubleshooting and end-user review.

1. **Routine Maintenance:** This sub-stage performs periodic maintenance of the software. As an example, routine maintenance coordinates the updates of the acquired software by upgrading it to a new version or by applying patches to it. It also performs pre-requisite upgrades, such as OS version and patches. Lastly, it documents all the maintenance works performed such as upgrades of software versions. It also updates the software knowledge base.
2. **Troubleshooting:** In this sub-stage, we solve routine software problem. Upon facing a problem such as a bug in the software, an user contacts the help desk. The help desk fixes the problem with the help of the vendor. A centralized helpdesk system with the concept of newsgroup or online problem submission makes the process efficient.
3. **End-User Review:** We review end-user requirements through periodic reviews. If there are unmet needs, the IT group should correct the vendor. The decision to replace the software with a new software is also met in this stage. If a replacement is needed software acquisition model is repeated.

13.4 Use of Database

In this stage, maintenance work such as upgrades, maintenance time, feedbacks etc are stored in the database.

13.5 Exit

Exit from this stage takes place after the software is replaced by a new software or disposed after the duration of its lifetime.

14. Conclusion

We have presented a general software acquisition life cycle model. The model begins with an initial user request and ends with maintenance and periodic requirements review. The model has two distinct phases. The first phase, called the "Choice Phase", iterates three main stages until a suitable COTS package is found. The second phase, called the "Implementation Phase", proceeds sequentailly. There are five main stages to the Implementation Phase. In addition to defining these stages, our model also specifices specific documents that flow through the model. These documents make up the inter-stage data flow.

Our Requirements, Analysis, and Presentation stages make use of a requirements weighting scheme. Our scheme allows the end users to weight their requirements. These weights are combined with testing scores to produce overall product grades. The grades are presented to the customer along with other parameters, such as estimated schedules and costs, such that the customer may make a buy/iterate decision.

In the Implementation Phase, we pay particular attention to process control. We monitor vendor and in-house product modification/customization. The deployment process also provides feedback to the acquisition team to monitor actual progress versus expected schedules.

Our model, taken as a whole, should provide a good process guide to a company implementing a large scale COTS acquisition. We hope that in the future, we will have the opportunity to put our model through field trials.

Appendix A. Process Example

A.1   Purpose

We give a simple example of using our process to choose a new game for the company network.

A.2   Procedure

The following sections take an example project through the Requirements, Analysis, and Presentation stages.
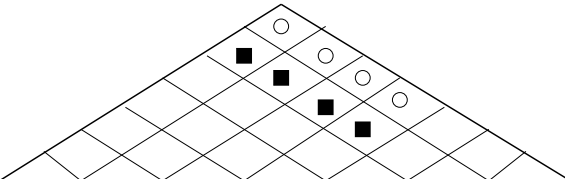
A.2.1   Requirements

Following section lists the requirements as determined by the first round of requirements elicitation. From the user descriptions, we have the following definitions for each requirement:

1. **Cool Graphics**: High quality graphics, rich in color and detail.
2. **Awesome Sounds**: High fidelity, stereo sounds.
3. **Cut Scenes**: Users like high quality video sequences between game scenarios.
4. **Multiplayer**: Users must play each other over the corporate network.
5. **Multiple Skill Levels**: We have players of many skill levels in the company. The game must accommodate this diversity.
6. **Quick Response**: The game must be responsive. Users did not give specifics, but they would "know it if they saw it."
7. **Multiple Scenarios**: A few users stated that they were concerned that they may grow tired of the game if it did not offer a selection of playing environments.
8. **Run on PC**: The IT group added a requirement that the game must run on IBM-style PCs, as that is the only platform in house.

The users also assigned weights to their requirements. The weights are shown in Figure A.1. They identified 3 "Must Have" requirements. The others were assigned an average of user stated preferences. Each Must Have requirement has a weight of "12".

After gathering the requirements, the IT group constructed the QFD table, as shown in Figure A.1. The IT group translated the first 4 requirements in to purely technical characteristics. These characteristics do not require user testing to verify. The IT group may verify these requirements by inspection or vendor query. The QFD table shows that there may be a conflict between the Multiplayer requirements and the Quick Response test for Single User. Likewise, it shows there is a strong relation between the Multiplayer requirements and the Quick Response Multiplayer requirement.

User requirements 5, 6, and 7 were translated in to qualitative technical requirements. Requirement 5 is a 1:1 mapping between user requirement and technical requirement. Requirement 6 was broken in to two technical requiments: Quick Response in Single Player mode (QR Single) and Quick Response in Multiplayer mode (QR Multi). Their test cases will surface testing distinctions. The IT group interpreted requirement 7 to mean "Interest Level" in the game. Again, the test case will reveal the exact interpretation.

TECH REQS. USER REQS.	24 Bits	3DFX	Hifi Stereo	Real Actors	IP	IPX	Dial Up	Runs Ok 10Mbps	MSL	QR Single	QR Multi	Interest Level	Weights
1	✕	✕											Must
2			✕										4
3	✕		✕	✕									2
4					✕	✕	✕	✕					Must
5									✕				2.5
6										✕	✕	✕	Must
7												✕	3.5

Figure A.1: QFD Table for Game

$$\text{1.UNAMBIGUITY } Q_{un} = \frac{n_{ui}}{n_r} = \frac{4}{7} = .57$$

$$\text{2.COMPLENESS } Q_{com} = \frac{n_A}{n_A + n_B} = \frac{6}{7} = .86$$

$$\text{3.CORRECTNESS } Q_{corr} = \frac{n_c}{n_r} = \frac{7}{7} = 1.00$$

Figure A.2: Quality Calculations for Game

As shown in Figure A.2, the IT group also performed a Quality calculation for these requirements. The IT group decided that requirements 5,6, and 7 were poorly defined and thus detracted from the Unambiguity rating. In an actual model implementation, a Q_{un} of 57% should cause the IT group to stop and revisit the user requirements. For the purposes of this example, we will press on.

For the Test Plan shown in Table A.1, we only show a full test case for "Multiple Skill Levels" user requirement. We also show how the "Quick Response" requirement was broken in to two test cases. The MSL requirement test MSL.1 defined the testers to be "Skilled game players" since the IT group thought that these users could best test for this quality. The QR requirement was broken in to two test cases QR.1 and QR.2. The IT group thought that active and infrequent game players would have a different view of what "Quick Response" meant. In our company of 30 users, 6 were "Active" and the other 24 were "Infrequent". Thus, we arrived at the 20/80 split.

A.2.2 Analysis

In the Analysis stage, our testers ranked four test cases as shown in Table A.2. After calculating the means and standard deviations as shown in table A.3, we see that Product X has a high deviation for test QRS.1 and that Product Y has a high deviation for tests QRS.2 and QRM.1. These deviations should cause the IT group to establish why the testers scored the products with high variance.

User Req	Tech Req	Test Case	TC Weight	TC Topic	TC Description
5	MSL	MSL.1	1.0	What Who How Grade	Does game difficulty progress with level? Skilled game players Play game over time How well did game keep you challenged?
6	QR single 0.5	QRS.1	0.2	What Who How Grade	Perceived response time Active game players Play game at different skill levels
		QRS.2	0.8	What Who How Grade	Infrequent game players
	QR multi 0.5	QRM.1	1.0	What Who How Grade	Perceived response time 1 Active, 6 Infrequent Play game at different skill levels

Table A.1: Game Test Cases

	Product X			Product Y			. . .
	tester1	tester2	tester3	tester1	tester2	tester3	
MSL.1	A	B+	B	B-	B	A-	. . .
QRS.1	A-	B	C	B+	B+	A	. . .
QRS.2	B+	B	B	C	A	B+	. . .
QRM.1	A-	A-	B+	C+	B+	A	. . .
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Table A.2: Example Product Grade Table

	Product X	Product Y	. . .
MSL.1	3.43±0.42	3.13±0.42	. . .
QRS.1	2.90±0.70	3.53±0.33	. . .
QRS.2	3.10±0.14	3.10±0.83	. . .
QRM.1	3.57±0.24	3.20±0.79	. . .
. . .	. . .	. . .	. . .

Table A.3: Example Product Grade Summary Table

A.2.3 Presentation

At the presentation stage of this example, we can calculate the scores for the technical requirements and then the user requirements based on the Product Grade Summary Table produced at the analysis stage. Table A.4 below shows the relationship between the technical requirements and the test cases: MSL produced one test case, MSL.1. QRS produced two test cases, QRS.1 and QRS.2, which were given the weights 0.2 and 0.8 respectively. QRM produced one test case,QRM.1.

	Product X			
	MSL.1 (1.0)	QRS.1 (0.2)	QRS.2 (0.8)	QRM.1 (1.0)
MSL	3.43±0.42			
QRS		2.90±0.70	3.10±0.14	
QRM				3.57±0.24
. . .	. . .	. . .	. . .	. . .
	Product Y			
	MSL.1 (1.0)	QRS.1 (0.2)	QRS.2 (0.8)	QRM.1 (1.0)
MSL	3.13±0.42			
QRS		3.53±0.33	3.10±0.83	
QRM				3.20±0.79
. . .	. . .	. . .	. . .	. . .

Table A.4: Example Relationship between Technical Requirements and Test Cases

Using the formulas below, we can calculate the scores for those technical requirements, which are shown in Table A.5. The calculation of the deviation is based on the formulas introduced in the presentation chapter.

$V_{MSL} = V_{MSL.1} * 1.0$

$V_{QRS} = V_{QRS.1} * 0.2 + V_{QRS.2} * 0.8$

$V_{QRM} = V_{QRM.1} * 1.0$

Table A.6 shows further relationship between the user requirements and the technical requirements. It can be seen that user requirement 5 produced MSL, while user requirement 6 produced QRS and QRM.

Using the similar formulas:

$V_{UR.5} = V_{MSL} * 1.0$

$V_{UR.6} = V_{QRS} * 0.5 + V_{QRM} * 0.5$

We can calculate the scores for the two user requirements. The results are shown in Table A.7.

	Product X	Product Y
MSL	3.43±0.42	3.13±0.42
QRS	3.06±0.20	3.19±0.67
QRM	3.57±0.24	3.20±0.79
. . .	. . .	. . .

Table A.5: Score for Each Technical Requirement on Each Product

	Product X		
	MSL (1.0)	QRS (0.5)	QRM (0.5)
5	3.43±0.42		
6		3.06±0.20	3.57±0.24
. . .	. . .	. . .	. . .
	Product Y		
	MSL (1.0)	QRS (0.5)	QRM (0.5)
5	3.13±0.42		
6		3.19±0.67	3.20±0.79
. . .	. . .	. . .	. . .

Table A.6: Example Relationship between User Requirements and Technical Requirements

	Product X	Product Y
5	3.43±0.42	3.13±0.42
6	3.32±0.16	3.20±0.52
. . .	. . .	. . .

Table A.7: Example Score for Each User Requirement on Each Product

We can also calculate the scores for other user requirements in this way. After we get the scores for all the user requirements, we can calculate the overall score for each product. In this example, if we assume that both product X and Y satisfy the requirements 1 to 4 well and got the full score 4 with deviation 0; and for requirement 7, both products got score 3 with deviation 0, then we can compute their scores using the weights assigned at the requirement stage (shown in Figure A.1). The final scores are shown in the following multiple metric graph. In this graph, which is used to compare the two products, we assume: X has the cost estimation of $1000, and schedule estimation of 10 days; Y has the cost estimation of $700, and schedule estimation of 15 days.

It can be seen from the graph that X has a higher overall score and a shorter implementation time, but it is also more expensive. Which product to choose depends on the main concerns of the acquirer.
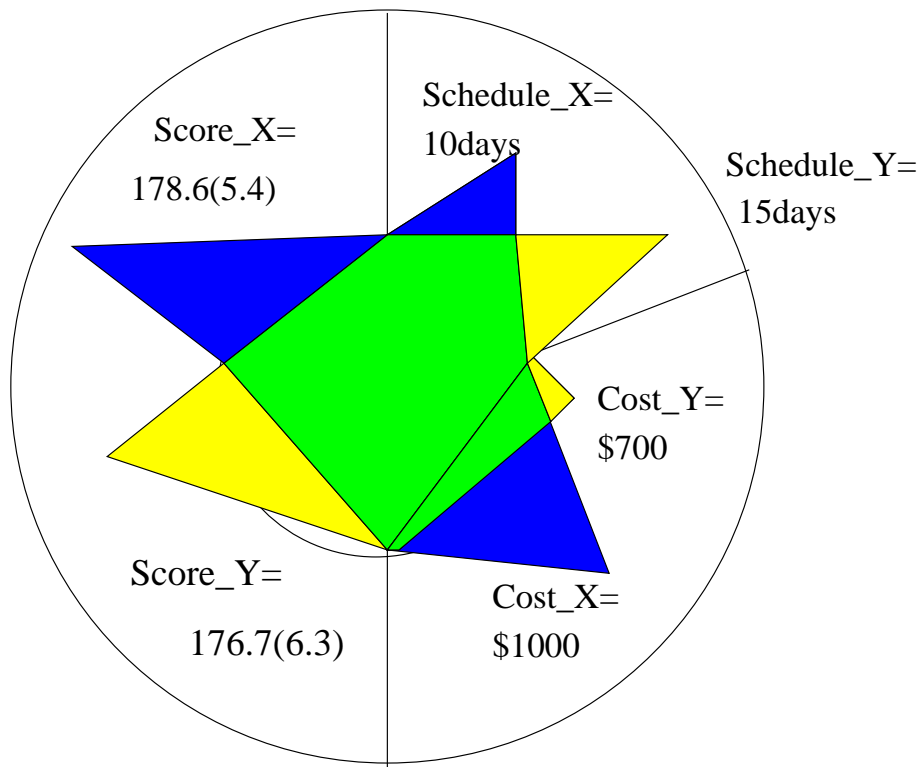
Figure A.3: Comparing Metrics of Two Products

References

[1] L. Alexander. *Selection Criteria for Alternative Software Life-Cycle Process Model.* MS-Theis, Fairfax, Virginia: Geroge Mason University, 1990.

[2] ANSI/IEEE Std 1042. *IEEE Guide to Software Configuration Management.* IEEE, New York, 1987.

[3] D.E. Avison, W.A. Eardley, and P. Powell. Suggestions for Capturing Corporate Vision in Strategic Information Systems. *Omega*, 26(4):443–459, Jul 1998.

[4] C. Beskow, J. Johansson, and M. Norell. Implementation of qfd:identifying success factors. In *Proceedings of IEMC*, pages 179–184. IEEE Computer Society, 1998.

[5] M. Beynon, B. Curry, and P. Morgan. The Dempster-Shafer Theory of Evidence: An Alternative Approach to Multicriteria Decision Modelling. *Omega*, 28(1):37–50, Jan 2000.

[6] B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy. Using the WinWin Spiral Model: A Case Study. *Computer*, pages 33–44, Jul 1998.

[7] Barry Boehm and Chris Abts. Cots integration: Plug and pray. *Computer*, pages 135–138, January 1999.

[8] B.W. Boehm. *Tutorial on Software Risk Management.* IEEE Computer Society Press, Los Alamitos, CA, 1989.

[9] D.T. Campbell and D. W. Fiske. Convergent and Discriminant Validation by the Multitrait-Multimethod Matrix. *Psychological Bulletin*, 56(2):81–105, Mar 1959.

[10] J. Chudge and D. Fulton. Trust and Co-operation in System Development: Applying Responsibility Modelling to the Problem of Changing Requirements. *Software Engineering Journal*, 11(3):193–204, May 1996.

[11] T.D. Clark Jr. Corporate Systems Management: An Overview and Research Perspective. *Communications of the ACM*, 35(2):60–75, Feb 1992.

[12] L. Cronbach. Coefficient Alpha and the Internal Structure of Tests. *Psychometrika*, 16:297–334, 1951.

[13] Alan Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, A. Ta, and M. Thefanos. Identifying and measuring quality in a software requirements specification. In *Proceedings of 1st Int'l Software Metrics Symp.*, pages 141–152, 1993.

[14] D.H. Gensch and S. Ghose. Differences in Independence of Irrelevant Alternatives at Individual vs Aggregate Levels, and at Single Pair vs. Full Choice Set. *Omega*, 25(2):201–214, Mar 1997.

[15] Charles Y. Glock. Survey Design and Analysis in Sociology. In Charles Y. Glock, editor, *Survey Research in the Social Sciences*, pages 1–62. Russell Sage Foundation, New York, 1967.

[16] J.A. Goguen and C. Linde. Techniques for Requirements Elicitation. In Richard H. Thayer and Merlin Dorfman, editors, *Software Requirements Engineering*, pages 110–122. IEEE Computer Society Press, second edition, 1997.

[17] Stephen Haag, M. K. Raja, and L. L. Schkade. Quality Function Deployment Usage in Software Deployment. *Communications of the ACM*, 32(1):42–49, Jan 1996.

[18] http://www.microsoft.com/smsmgmt.

[19] http://www.networkd.com/landesk/.

[20] M.I. Hwang and R.G. Thorn. The Effect of User Engagement on System Success: A Meta-Analytical Integration of Research Findings. *Information and Management*, 35(4):229–236, Apr 1999.

[21] IEEE Std 1028. *IEEE Standard for Software Reviews and Audits*. IEEE, New York, 1989 corrected edition, 1989.

[22] IEEE Std 1062. *IEEE Recommended Practices for Software Acquisition*. IEEE, New York, 1998.

[23] IEEE/EIA Std J-STD-016-1995. *Software Development Acquirer-Supplier Agreement*. IEEE, New York, 1995.

[24] R. Jain. Key Constructs in Successful IS Implementation: South-East Asian Experience. *Omega*, 25(3):267–284, Jun 1996.

[25] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., New York, 1991.

[26] K.B. Keeling and R.J. Pavur. Relationships Between Cronbach's Reliability Estimate and Research Design Characteristics in Information Systems. http://www.sbaer.uca.edu/docs/proceedingsII/98dsi0855.htm, Nov 1998.

[27] Geoffrey Keppel. *Design and Analysis. A Researcher's Handbook*. Prentice-Hall, Englewood Cliffs, New Jersey, second edition, 1982.

[28] David L. Lanning and Taghi M. Khoshgoftaar. The impact of software enhancement on software reliability. *IEEE Transaction on Reliability*, 44(4):60–66, Dec 1995.

[29] James P. Lewis. *Mastering Project Management*. McGraw-Hill, New York, 1998.

[30] U. Lindqvist and E. Jonsson. A Map of Security Risks Associated With Using COTS. *Computer*, 31(6):60–66, Jun 1998.

[31] T. Menzies, S. Easterbrook, B. Nuseibeh, and S. Waugh. An Emperical Investigation of Multiple Viewpoint Reasoning in Requirements Engineering. In *Proceedings IEEE International Symposium on Requirements Engineering*, pages 100–109. IEEE Computer Society, Jun 1999.

[32] P. Nelson, W. Richmond, and A. Seidmann. Two Dimensions of Software Acquisition. *Communications of the ACM*, 39(7):29 – 35, 1996.

[33] S.L. Pfleeger, J.C. Fitzgerald Jr., and D.A. Rippy. Using multiple metrics for analysis of improvement. *Software Quality Journal*, 1:27–36, 1992.

[34] C. Potts and W.C. Newstetter. Naturalistics Inquiry and Requirements Engineering: Reconciling Their Theoretical Foundations. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, pages 118–127. IEEE Computer Society Press, Jan 1997.

[35] S. Rivard, P. Lebrun, and J. Talbot. Measuring the Quality of User-Developed Applications. *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, 4:471–478, Jan 1991.

[36] J. Rumbaugh. Getting Started Using Use Cases to Capture Requirements. In Richard H. Thayer and Merlin Dorfman, editors, *Software Requirements Engineering*, pages 123–127. IEEE Computer Society Press, second edition, 1997.

[37] P. Sawyer, I. Sommerville, and S. Viller. Capturing the Benefits of Requirements Engineering. *IEEE Software*, 16(2):78–85, Mar-Apr 1999.

[38] Geri Schneider and Jason P. Winters. *Applying Use Cases A Practical Guide.* Object Technology Series. Addison-Wesley, Reading, Massachusetts, 1998.

[39] Michael Scriven. Product Evaluation. In Nick L. Smith, editor, *New Techniques for Evaluation*, volume 2, pages 121–166. Sage Publications, Beverly Hills, 1981.

[40] H. Shin and J. Lee. A Process Model of Application Software Package Acquisition and Implementation. *Journal of Systems and Software*, 32(1):57–64, Jan 1996.

[41] Ian Sommerville and Pete Sawyer. *Requirements Engineering – A Good Practice Guide.* John Wiley & Sons, New York, 1997.

[42] C.R. Symons. Function Point Analysis: Difficulties and Improvements. *IEEE Transactions on Software Engineering*, 14(1), Jan 1988.

[43] J.T.C. Teng, K.D. Fiedler, and V. Grover. An Exploratory Study of the Influence of the IS Function and Organizational Context on Business Process Reengineering Project Initiatives. *Omega*, 26(6):679–698, Nov 1998.

[44] T. Teo and W.R. King. An Assessment of Perceptual Differences Between Informants in Information Systems Research. *Omega*, 25(5):557–566, Sep 1997.

[45] Vu Tran and Dar-Biau Liu. A risk-mitigating model for the development of reliable and maintainable large-scale commercial-off-the-shelf integrated software systems. In *Proceedings of Annual Reliability and Maintainability Symposium*, pages 361–367. Annual Reliability and Maintainability Symposium, Philadelphia, PA, USA, 13-16 Jan. 1997, 1997.

[46] Karl E. Wiegers. *Software Requirements.* Microsoft Press, Redmond, Washington, 1999.