SOFTWARE QUALITY ANALYSIS: A VALUE-BASED APPROACH

by

LiGuo Huang

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

December 2006

Dedication

To my dad Rongxin Huang, my mom Ming Qian

To my husband Rong Huang

Acknowledgements

This dissertation research would not have been possible if there were not the contributions of many hearts and minds over years. Foremost, I thank my advisor, Dr. Barry Boehm, for his invaluable guidance, support and enthusiastic encouragement. During my Ph.D. studying in Center for Software Engineering at the University of Southern California, Dr. Boehm has always been an excellent and dependable mentor in helping me achieving my research and professional goals. Both his diligence and insightfulness in academic research and his personality will always be an exemplary role model in my life. I also want to express my thanks to other members in my qualifying and defense committees, Dr. Ahmed Helmy, Dr. Ming-deh Huang, Dr. Kai Hwang and Dr. Nenad Medvidovic, for their valuable comments and words of encouragements. Many thanks to my USC-CSE officemates, labmates and classmates for sharing with me the joys and sorrows of the Ph.D. student life.

A special thank is given to my VBSQA-OPN research collaborators Hao Hu, Jidong Ge, Jian Lü and students (Weijie Zhu, Qiang Lin, Chao You and Fei Xiong) in Institute of Computer Software at Nanjing University for their efforts in tool prototyping. Special thanks are also extended to DIMS project team in Neusoft Co., Ltd. for data supply and valuable feedback.

Finally, from my heart, I would like to thank my parents for their love and encouragement. They have been inspired me to always strive for excellence and furnished me the curiosity of learning. I also want to thank my husband, Rong for his unconditional support during these challenging years.

Table of Contents

List of Tables

List of Figures

Abstract

Quality is a major challenge for all complex software systems. Some important attributes of software quality include reliability, availability, safety, security, survivability, performance, accuracy, etc. These have long been requirements of aerospace and defense systems. Now, equally challenging requirements are being placed on “everyday systems” that increasingly provide the infrastructure for our daily lives such as commercial, e-business and embedded systems. They are subject to modest, usually tacit, often stakeholder-specific quality requirements. And it is important that we can define and meet a software system’s quality requirements to be fit for its purpose.

However, there are many views of software quality. And there also exist many ways to suboptimize its achievement and/or misallocate limited project resources using incomplete views. In addition, complex missions or projects usually involve a large and heterogeneous group of stakeholders with various (often time-varying) quality priorities and different (often conflicting) needs. This makes both one-size-fits-all quality metrics and software development processes driven by such metrics risky to use. This also points to the need for better frameworks to define, reason about and achieve quality attributes.

Based on these observations, the primary goal of this dissertation is to develop a Value-Based Software Quality Analysis framework that integrates the stakeholder/value approach into quality attribute definitions, metrics, models and development processes aiming at achieving the appropriate quality levels for software systems. This framework pays explicit attention to business values that a software system generates for its success-critical stakeholders. It helps us to understand the nature of quality and to achieve the stakeholder mutually satisfactory quality requirements. It addresses the above problems in four aspects: 1) value-based definitions of software quality attributes; 2) value-based quality metrics; 3) Value-Based Software Quality Model (VBSQM) to reason about the Return On Investment (ROI) of quality and to perform combined risk analyses; and 4) Value-Based Software Quality Achievement (VBSQA) process.

Finally, the VBSQM and VBSQA process have been applied and found to be effective on three diverse software systems with different value profiles: a USC-CSE e-service project, the NASA/USC Inspector SCRover project, and a real-world ERP software development project in industry.

Chapter 1

Introduction

1.1 Evolving Views of Software Quality

The early institutional focus on software quality was primarily initiated by the U.S. Department of Defense (DoD), where software quality is one of the most critical concerns of mission-critical defense systems. Based on the requirements-driven, contract-oriented waterfall-model software development, its major 1974 standard, MIL-S-52779, "Software Quality Assurance Program Requirements" [DoD 1974], defined the objective of software quality assurance (QA) as, "to assure that the software delivered under the contract meets the requirements of the contract."

The major pitfall of this approach is that QA is purely based on the initial contract. If the contract specifies poor or incomplete quality requirements, you will get poor quality software. This happened to DoD and commercial organizations in numerous ways such as specifying poor user interfaces, specifying requirements obtained from the wrong users, getting unmaintainable software by missing maintenance and diagnostic requirements.

In the 1980's, there began a trend away from the 1970's contract-oriented specification compliance toward service-oriented customer satisfaction as the primary quality objective. Software quality assurance approaches such as Total Quality Management (TQM) [Deming 1989] and Quality Function Deployment (QFD) [Eureka-Ryan 1988] based quality on expectations of customers, who were generally interpreted to include users of software products. Thus, the 1990 definition of “quality” in the IEEE Standard Glossary of Software Engineering Terminology [IEEE 1990] added “… meets customer or user needs or expectations” to its earlier definition of “…meets specified requirements.”

The major difficulty with the customer-satisfaction approach is that customers often lack a complete view of tradeoffs and interactions among their concerned software quality attributes. And they often neglect other quality attributes such as maintainability which they are indirectly dependent on. For instance, customers have pushed overly ambitious performance objectives which led to unaffordable and/or unmaintainable software systems, or have pushed to adopt a poorly-architected prototype with nice usability features but poor scalability, availability, and/or interoperability.

Initiatives to address these problems in the 1990’s have focused on identifying a full set of success-critical stakeholders in a software system and pursuing the objective of negotiated stakeholder win-win relationships among software quality attributes. This expands the scope of “quality” to include the proposition of system acquirers on cost or affordability, the proposition of the software maintainer on maintainability and scalability, the proposition of stakeholders of interoperating systems on interoperability, and others such as the proposition of the general public on safety or privacy. This has led to new organizational approaches such as Integrated Product Teams, expanded versions of QFD [Pardee 1996], and process approaches such as the WinWin Spiral Model [Boehm et. al. 1995a].

1.2 Future Trends and Software Quality Challenges

The future trends of software system development include:

- **An increased emphasis on stakeholders and end value**

With the increasing emphasis on success-critical stakeholder value in software system development and the need for rapidly evolving systems, software quality definitions, assessment models and achievement processes need to address the different value propositions of various stakeholder classes.

- **Increasingly rapid change**

When added to the trend toward emergent systems requirements, the pace of change places a high priority on systems and software engineering process agility and investments in continuous learning for both people and organizations [Boehm 2005].

The major challenges on software quality assessment and achievement implied by the future trends of software development are as follows:

1. **The universal one-size-fits-all software quality metrics are unachievable in most project situations.**

Value dependencies vary significantly by stakeholders and situations, making statements such as “Your system has a software reliability rating of 0.613” usually meaningless. Occasionally, a very stable organization can develop and manage to an organization-specific software reliability metric whose change from 0.604 to 0.613 or from 0.621 to 0.613 will be highly meaningful. But in most situations, stakeholder and situation differences make such single software quality metrics infeasible.

2. **Stakeholder value dependencies on software quality attributes are often in conflict and require negotiated solutions.**

Figure 1.1 illustrates this situation with respect to four of the primary stakeholder classes in information system acquisition and development: users, acquirers, developers and maintainers. It summarizes the results of our analysis [Al-Said 2003, Boehm et al. 2000b, Boehm 1999] of a number of failed projects in [Flowers 1996, Glass 1998]. It shows that the value dependencies of the four most common stakeholder classes in software development (users, acquirers, developers, maintainers) are often in conflict, and demonstrates that the need to reconcile different stakeholders' value dependencies is built into the nature of software development projects.

The red or gray lines in Figure 1.1 show the specific conflicts for one of the failed projects: the Bank of America Master Net trust management project. The users' value propositions included many features (3.5 million lines of code when the project was cancelled) and high levels of performance, reliability, and availability. The acquirers' value propositions included a limited development budget and schedule ($20M and 9 months to an initial operational capability). The selected developers' bid was based on the value proposition of reusing software from successful small trust management systems they had developed on Prime computers.

The resulting solution turned out to have major value conflicts with the users' number of features and need for early capabilities and the acquirers' limited budget and schedule: the project was cancelled when no adequate capability was in sight after the expenditure of $88M and 48 months. It also had value conflicts with users on high levels of service or quality attributes (slow response time, frequent crashes) and with the users and maintainers on applications compatibility and ease of transition and maintenance (Bank of America was an IBM mainframe shop).

A number of other value dependency conflicts or success model clashes are also evident from Figure 1.1. The main points are that these are built into software project environment from the beginning, and that they represent a set of value-based issues that need to be considered by the project's quality assurance approach.

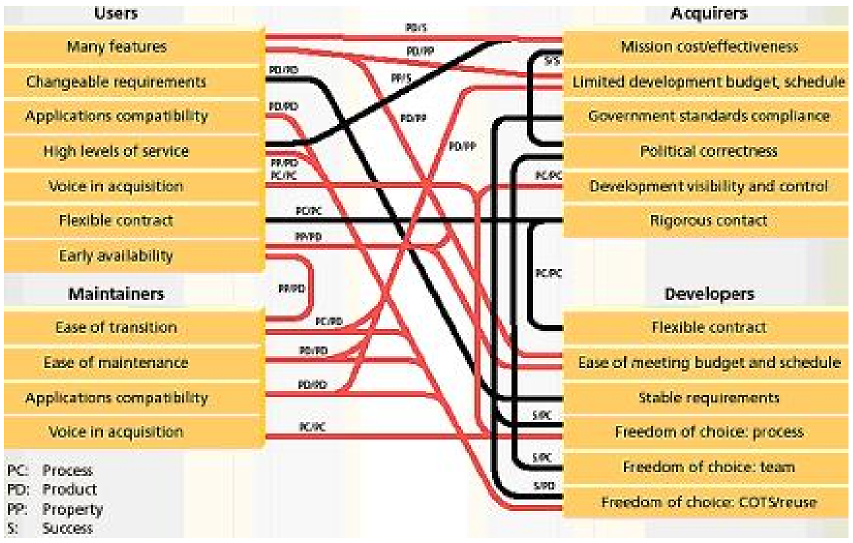


Figure 1.1. Built-in Stakeholder Value Conflicts. The Red or Gray Lines Show Conflicts from the MasterNet System.

3. **Stakeholders have often-emergent, time-varying priorities for software quality attributes.**

Even for the same person in the same organization, stakeholders' value dependencies are likely to change with time to reflect their Maslow need hierarchy (see Figure 1.2), in which unsatisfied lower-level needs dominate higher-level needs, but in which satisfied lower-level needs are no longer motivators [Maslow 1954]. Thus, for example, the corporate survival of a startup company (the equivalent of food and drink for its leaders) will often depend initially on novelty and usability to attract early adopters, but if the company is successful and builds an increasing installed base of more mainstream customers, it may next find that its top priority becomes performance and scalability rather than more novelty and usability. As customers increasingly depend on the system's operation, system reliability and availability will dominate further improvements in performance; and eventually further investments in reliability and availability may be dominated by investments in security, as threats and vulnerabilities from information system attackers become more significant.

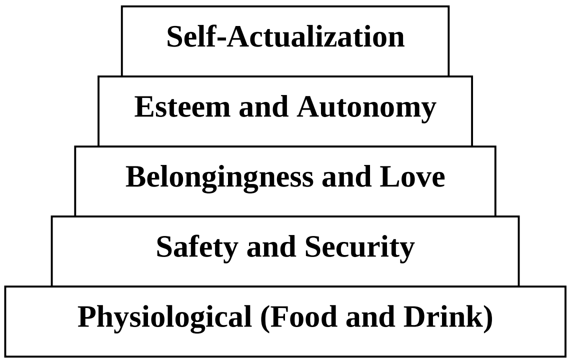


Figure 1.2. Maslow Human Need Hierarchy (A. H. Maslow, *Motivation and Personality*, 1954)

4. **Static, "snapshot" optimizations among stakeholder value dependencies may have short solution lifetimes.**

Many multi-criterion decision solutions involve applying stakeholder-determined weights to software quality attributes and choosing the solution that maximizes a weighted combination of alternative solutions' current quality attribute values [Gilb 1969, CODASYL 1976, Saaty 1989]. If the stakeholders are in time-varying priority situations, the optimality of the initial solution needs to be monitored and adjusted as priority and value dependencies change.

5. **Multi-criterion decision solutions are complicated by tradeoff relations among software quality attributes.**

Many software quality attributes reinforce each other. An unreliable system will not be very secure; a poorly-performing system will not be very usable. On the other hand, many conflicts arise in trying to satisfy multiple quality criteria. Complex security defenses slow down performance and hamper usability. Fault tolerance solutions spread information around and complicate security defenses, along with adding performance overhead. Tightly-coupled performance optimizations complicate the achievement of evolvability and reusability, as well as aspects of security and fault tolerance. All of the above add to project costs and schedules. These tradeoff relations complicate the ability to find solutions that satisfy a combination of quality attribute levels.

6. **Stakeholder/value-oriented metrics help avoid measurement dysfunction, and help steer a project toward stable win-win stakeholder incentive structures.**

Oversimplified project management metrics tend to lead project performers into measurement dysfunction (gaming the metrics) and unsatisfactory project outcomes [Austin 1996]. Examples in the quality area include the conflicting liveness and performance requirements from different stakeholders in NASA Earth

Observation System Distributed Information System (EOSDIS) example discussed in the section 3.1.2.2; committing projects to overambitious performance goals [Boehm 2000] or cost and schedule goals [Standish 1995, 2001]; and failed projects such as the Master Net project discussed in item 2 above. Mixed quantitative and qualitative management metrics approaches that capture a more complete set of stakeholder values such as the Balanced Scorecard technique [Kaplan-Norton 1996] are more likely to succeed.

7. **There is a need for better value estimating relationships for software quality attributes.**

For example, the information broker e-trade.com valued an hour of downtime as a cost of $8000 in revenues. Ideally, one would like to know at least the shape of such value/utility functions for various stakeholder/value dependencies.

1.3 Problem Statement

This dissertation will address the problem of software quality modeling and achievement process using the value-based approach. It will also examine the value-based definitions of software quality attributes and applications of the model and process in real-world projects. A concise statement of the problem is:

What are the effective models and processes to determine the appropriate levels of software quality investments and to help achieve stakeholder WinWin-balanced software quality outcome?

In support of addressing the central problem of constructing the value-based software quality model and achievement process, there are several sub-problems that need to be answered as following:

- How to define the software quality attributes from stakeholder/value perspectives?
- How to measure the achievement of software quality attributes?
- How to determine how much software quality investment is enough?
- How to use the value-based software quality definition, metrics and model to drive the software development processes and to help project success-critical stakeholders define, negotiate, develop, monitor and control the achievement of the mission-specific combinations of quality attributes?

1.4 Dissertation Approach and Hypotheses

This dissertation defines a framework for Value-Based Software Quality Analysis to solve the above problems in four aspects:

- Value-based definitions of software quality attributes
- Value-based software quality metrics
- Value-Based Software Quality Model (VBSQM)
- Value-Based Software Quality Achievement (VBSQA) process

Three hypotheses based on the framework are proposed and validated in this dissertation:

- **Hypothesis 1**: Value-Based Software Quality Model (VBSQM) can be used to determine how much software quality investment is enough in different value situations.
- **Hypothesis 2**: Assuming non-linear value functions (e.g., Pareto distribution) are used, value-based software quality achievement techniques improve project return on quality investments and reduce the overall project risks.
- **Hypothesis 3**: The Value-Based Software Quality Achievement (VBSQA) process can be applied by rational decision-makers to
    - **3a.** determine whether a software system with stakeholder mutually satisfactory software quality (Q-) attribute requirements is achievable.
    - **3b.** help stakeholders and projects avoid software Q-attribute mismatches and achieve successful software quality outcomes.

Figure 1.3 illustrates our research methodology in answering the thesis questions. We started with reviewing the existing literature that addresses software quality related research including definitions, metrics, models and process strategies. Then we identified the problems and challenges in software quality modeling and achievement processes. To address those problems and challenges, a software quality analysis framework was proposed using value-based approach. Two major components in this framework are Value-Based Software Quality Model (VBSQM) and Value-Based Software Quality Achievement (VBSQA) process. To validate the ROI analysis capability of VBSQM, we first applied it to two different types of software applications and then to different mission scenarios in one software project. To validate the combined risk analysis capability of VBSQM, we applied it to three representative project business cases in industry. To evaluate the VBSQA process, we first experimented it using two USC-CSE project case studies including one collaborative testbed project with NASA/JPL for High Dependability Computing Program (HDCP). Then we applied it to a real-world Enterprise Resource Planning (ERP) software development project in China. Based on the real-world experience, we refined and formalized the process to improve its future applications.

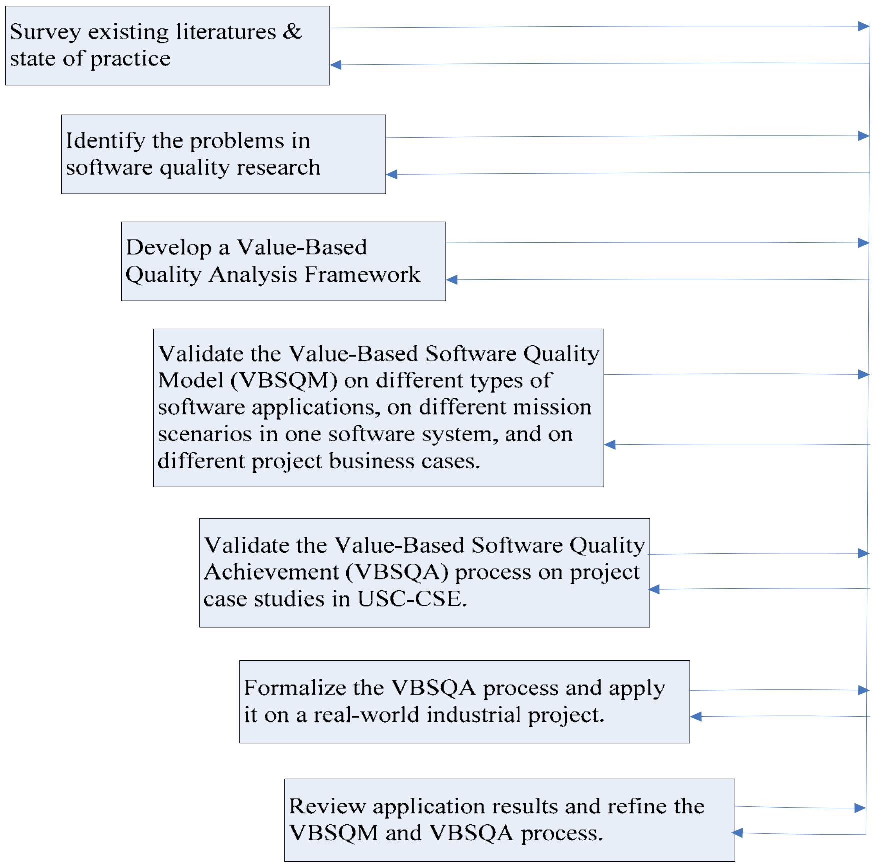

Survey existing literatures & state of practice
Identify the problems in software quality research
Develop a Value-Based Quality Analysis Framework
Validate the Value-Based Software Quality Model (VBSQM) on different types of software applications, on different mission scenarios in one software system, and on different project business cases.
Validate the Value-Based Software Quality Achievement (VBSQA) process on project case studies in USC-CSE.
Formalize the VBSQA process and apply it on a real-world industrial project.
Review application results and refine the VBSQM and VBSQA process.

Figure 1.3. Research Methodology

1.5 Dissertation Outline

The rest of this dissertation is organized as follows. Chapter 2 summarizes the related work on software quality definitions, metrics, models, assessment frameworks, process strategies, and value-based software engineering research with its addressal on software quality challenges. Chapter 3 presents the Value-Based Software Quality Analysis Framework including value-based definitions on quality attributes, value-based quality metrics, VBSQM, and VBSQA process. Chapter 4 shows the applications of VBSQM to validate the Hypothesis 1 and 2. Chapter 5 describes the applications of VBSQA process on two USC-CSE projects and on the real-world ERP software development in China. Hypothesis 3 a and b are validated by the project results and application feedback. Chapter 6 discusses the key contributions of this dissertation research and future research challenges in this area.

Chapter 2

Related Work

The major challenges of the research include the lack of a well-formalized and tangible definition of software quality and stakeholder/value-relevant software quality metrics and models for project decision-makers. Furthermore, how to use the metrics and models to drive the development processes on software quality achievement has not been sufficiently explored either. Based on these research challenges, this dissertation research expands over software quality definitions, metrics and model, and process strategies, which are then integrated into a software quality analysis framework. We have studied a variety of related research in each field. Section 2.1 through 2.3 summarizes the related work in each field. Section 2.4 summarizes the related work in the arena of value-based software engineering and its addressal of software quality challenges.

2.1 Software Quality Attributes and Relationships

2.1.1 Definitions of Software Quality and Quality Attributes

Many definitions of software quality have been proposed in the literature. They can be categorized into two classes. One class includes the traditional definitions of each attribute of software quality as an independent sub-discipline of software quality. In this dissertation, we only summarize the definitions of a few of the attributes (Reliability, Safety) as the example.

The definitions of Reliability can be found in the literatures such as IEEE Standard Glossary of Software Engineering Technology [IEEE 1990], IEEE Standard Dictionary of Measures to Reliable Software (IEEE Std 982.1-1988) [IEEE 1988a, 1988b], Handbook of Software Reliability Engineering [Lyu 1996], IEEE Standard for a Software Quality Metrics Methodology [IEEE 1992], etc. An example of such definition of *Reliability* is the ability of a system or component to perform its required functions under stated conditions for a specified period of time [IEEE 1990]. More definitions of Reliability and its factors are covered in [Rus et. al. 2003].

In [Leveson 1995], Safety is defined as *"freedom from accidents or losses"*. Software system safety implies that the software will execute within a system context without contributing to hazards. A hazard is defined as a state or set of conditions of a system (or an object) that, together with other conditions in the environment of the system (or object) will lead inevitably to an accident (loss event). Safety-critical software is any software that can directly or indirectly contribute to the occurrence of a hazardous system state. Safety-critical functions are those system functions whose operation (correct, incorrect, or lack of operation) could contribute to a system hazard. Other definitions of Safety can be found in the IEEE 1228-1994 (Software Safety Plans) [IEEE 1994], IEEE 1219-998 (Software Maintenance) [IEEE 1998].

Another class is the integrative definition of software quality as dependability, which our research is based on. Avizienis, Lapire and the IFIP (International

Federation for Information Processing) working group 10.4 on dependability computing and fault tolerance proposed what is probably the most well known collection of dependability concepts and definitions [IFIP WG10.4, Avizienis et. al. 2002, Lapire 1992]. The IFIP WG 10.4 defines dependability as *the property of a computer system such that reliance can justifiably be placed on the service it delivers* [IFIP WG10.4]. This definition first relates dependability to reliance on user services. [Avizienis et. al. 2002] further defines dependability as a composite concept consisting of several attributes and they present their definition of dependability using a dependability tree shown in Figure 2.1. Depending on the application(s) intended for the system, different emphasis may be put on different facets of dependability which are viewed as the attributes of dependability (i.e., availability, reliability, safety, confidentiality, integrity and maintainability). There are three implications from this definition. First, the attributes of dependability enable the properties which are expected from the system to be expressed. Second, the impairments to dependability include faults, errors and failures. Certain impairments occur when the delivered service deviate from fulfilling the expected system functions. Third, to ensure the dependability of a software system requires the combined utilization of certain engineering means which include fault prevention, fault tolerance, fault removal and fault forecasting. The major strength of the dependability concept is its integrative nature, which enables to put into perspective the more classical notions of reliability, availability, safety, security, maintainability, etc., which are now seen as attributes of dependability [Laprie 1998].

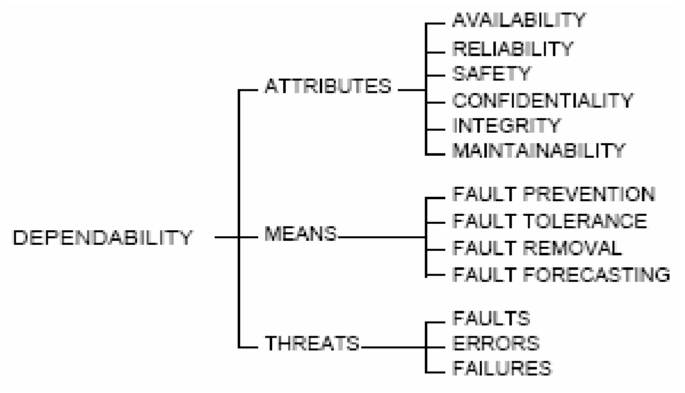


Figure 2.1. The Dependability Tree

Other related work on software quality attributes includes the TRW Characteristics of Software Quality study performed for the National Bureau of Standards [Boehm et. al. 1978] which provides a general framework for reasoning about software quality attributes; Gilb's work on quality attribute specification and management [Gilb 1976, Gilb 1988]; and the Fraunhofer-Maryland series of dependability attribute reports [Rus et. al. 2003].

Firstly, the definitions we summarized above are mostly general and qualitative. They are difficult to be applied and operated on realistic projects. To be operational, a definition needs to be strictly related to the specific context it refers to (the project scenarios and stakeholders). Secondly, they are mostly value-neutral and stakeholder irrelevant. An example was provided at one point of the operation of the NASA Earth Observation System Distributed Information System (EOSDIS). The system operators were incentivized to optimize the system for liveness, which often left the users with 2-week response times to queries. An example lose-lose user response is to flood the system with whatever queries one might need results for at some point. In this case, managing to two different Reliability numbers, one for liveness probability and another for (say) 10-minute response time probability, would be better. And the alternative is to have a single value of Reliability that is irrelevant or misleading to most of the stakeholders.

2.1.2 Relationships among Software Quality Attributes

[In 1998] maps common stakeholders' primary concerns onto quality attributes as shown in Figure 2.2. It also proposes an approach to identify and resolve the conflicts among quality attributes based on product and process strategies. This dissertation research extends the mapping by including a more comprehensive set of stakeholder classes and quality attributes. It also proposes a value-based quality model to perform tradeoff analysis among conflicting quality attributes and a value-based process to achieve stakeholder mutually satisfying software quality requirements.

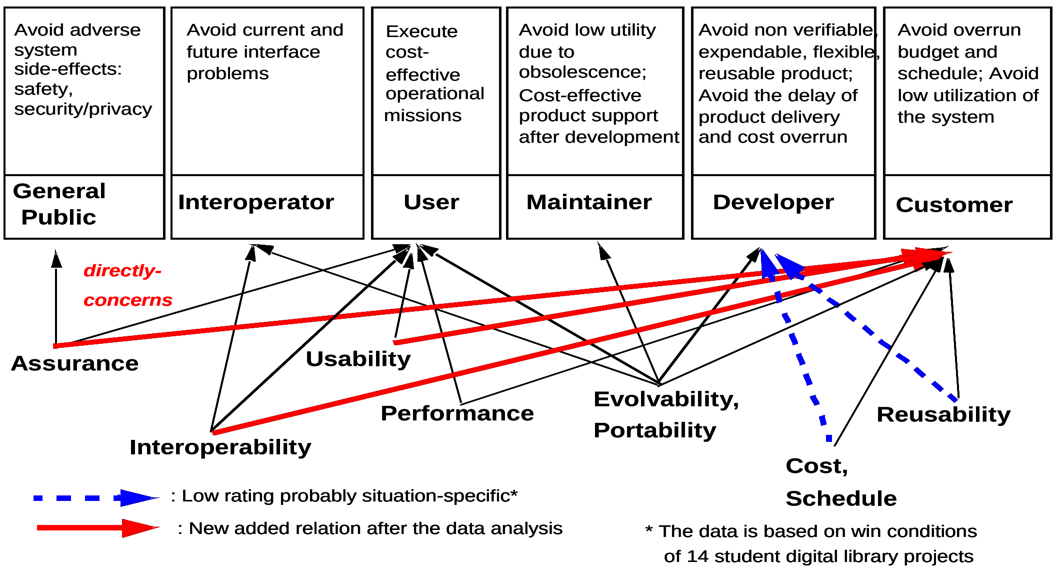

Avoid adverse system side-effects: safety, security/privacy
General Public
Avoid current and future interface problems
Interoperator
Execute cost-effective operational missions
User
Avoid low utility due to obsolescence; Cost-effective product support after development
Maintainer
Avoid non verifiable, expendable, flexible, reusable product; Avoid the delay of product delivery and cost overrun
Developer
Avoid overrun budget and schedule; Avoid low utilization of the system
Customer
directly-concerns
Assurance
Usability
Performance
Interoperability
Evolvability, Portability
Reusability
Cost, Schedule
: Low rating probably situation-specific*
: New added relation after the data analysis
* The data is based on win conditions of 14 student digital library projects

Figure 2.2. Empirically First-Order Mapping of Stakeholders Primary Concerns onto Quality Attribute

2.2 Software Quality Metrics, Assessment Frameworks and Models

2.2.1 Software Quality Metrics

Reliability is an attribute of software quality. Traditional ways to measure Reliability falls into two classes: 1) Time-based definitions; 2) Domain-based definitions that are also dependent on the applications. The examples of the first class are:

- Failures per time unit (failure intensity or failure rate)
- Mean Time To Failure (MTTF)
- Mean Time Between Failures (MTBF)
- The probability that the software will run for a given period of time without failure.

The examples of the second class are:

- failures/1000 pages printed (Hewlett Packard for printers)
- failures/1000 transactions (for sales, bank and ATM systems)
- failures/1000 calls for telecommunication systems (for telephone companies)

Other metrics that support reliability assessment are: CPU time, elapsed execution time, number of test runs, number of transactions, and number of failures of different classes. Similarly, one of the typical ways to measure Performance is system response time. An example is the system should response within 3 minutes.

There are some limitations with such metrics for software quality attributes:

- Not all failures have the same impact; some of them can cause just a minor inconvenience, while others could be mission critical or life critical; the impact of a failure depends on the failure frequency and failure severity.
- They are value-neutral and scenario-independent so that they tend to be one-size-fits-all metrics for a software system. In practice, such one-size-fits-all metrics are not achievable as discussed in section 1.2.

2.2.2 Software Quality Assessment Frameworks and Models

The classical reliability models summarized in [Lyu 1996] include the follows:

- Exponential Failure Time Class of Models
  - Jelinski-Moranda De-eutrophication Model
  - Nonhomogeneous Poisson Process (NHPP) Model
  - Schneidewind’s Model
  - Musa’s Basic Execution Time Model
  - Hyperexponential Model
- Weibull and Gamma Failure Time Class of Models
  - Weibull Model
  - S-Shaped Reliability Growth Model
- Infinite Failure Category Models
  - Duane’s Model
  - Geometric Model

- Bayesian Models
    - Musa-Okumoto Logarithmic Poisson
    - Littlewood-Verrall Reliability Growth Model

The above reliability models assume that each failure and defect is equally important.

Based on the integrative definition of software quality as dependability, Koopman and Maderia describe the problem of dependability benchmarking and prediction as a grand challenge in [Koopman 1999]. One of the two challenges they propose is to be able to predict the dependability of a system before it is deployed. And the dependability benchmark suites should include specifications, measures, workload, fault load, instrumentation, procedures and rules.

Madeira and Koopman in [Madeira 2001] try to develop a benchmarking framework for dependability evaluation based on the IFIP WG 10.4 dependability exploration. They intend to develop a framework integrating the measurement techniques and the collection of key components of dependability evaluation, which allows the quantification of dependability attributes or the characterization of the systems in well-defined dependability classes. Their dependability benchmarking dimensions include factors such as product vs. process, life cycle phases, user perspective, workload, upsetload, etc. They aim to provide a uniform, repeatable and comparable way of evaluating the behavior of components and computer systems in the presence of faults.

Wilson et. al. in [Wilson 2002] seek to enable comparison of different computer systems in the dimensions of availability, data integrity, disaster recovery and security. They develop a dependability comparison framework using different criteria for different applications. Their application types include transaction processing, message handling, process control, analytical types, and search and retrieval. Furthermore, they define classes for each application type whose boundaries are based on the natural breakpoints in the spectrum of user-perceived requirements. The highest class is always perfect behavior, whether or not it's achievable with current technology.

Arlat in [Arlat 2001] describes a benchmarking framework specification for the availability attribute of software quality. The specification distinguishes five system classes: transparent to user, retryable workload, delayable workload, reduced impact of failure, and nothing special. They apply a collection of availability factors with various criteria within each system class. The proposed criteria (minimum standards, required disclosures and comparative measurements) should meant not only to characterize a target system (belonging to a given class) with respect to specific availability factors; but also to be testable. And each criteria has its own evaluation method.

Rus et. al. in [Rus et. al. 2002] propose a series of steps to evaluate software quality achievement technologies aiming at assessing and improving software quality.

Other related work on software quality metrics and assessment frameworks includes the GE Software Quality Metrics study for the Air Force [McCall et. al. 1977] which incorporates 11 criteria encompassing product operation, product revision, and product transition; the European COQUAMO project [Kitchenham 1989]; IBM's CUPRIMDSO approach [Radice 1985]; Hewlett Packard's FURPS approach [Grady 1992a, 1992b]; the CMU Software Engineering Institute's Architecture Attribute Analysis program [Bass 1998].

All the above assessment frameworks and processes are only targeting at evaluating or comparing the quality of software systems. Part of our goal is similar, but we expand the scope beyond software quality measurement by using the software quality metrics to drive the development process. Our framework aims to find the optimal level of software quality investments based on the stakeholders' value propositions; and to achieve/monitor and control a software system's quality through the cost-benefit analysis of quality attributes. Thus our software quality analysis framework covers the following four aspects: quality attribute definitions, metrics, cost-benefit analysis model and process strategies.

The following two are the most compatible and complementary with our research objectives in that both of them argue about the importance of stakeholder involvement into the dependability assessment.

Huynh et al. in [Huynh 2003] propose a Center of Mass (COM) model to represent their view of software quality/dependability as a multi-attribute and multi-stakeholder concept. They use utility to represent the values of quality attributes, weights to represent the importance of an attribute to a stakeholder through stakeholder survey. And they add "confidence" to the stakeholders' answers. Then they can draw the COM model for each stakeholder's quality needs and evaluate whether the current system quality satisfies all the stakeholders' expectations. However, it doesn't provide a solution to avoid the one-size-fits-all metrics for stakeholders to define their software quality requirements.

The Unified Dependability Model (UMD) proposed in [Basili 2004] aims to establish a common language for discussing a variety of software quality/dependability attributes and to make them measurable. Stakeholders specify their needs by identifying, for the system or a specific service (scope), potential failures and hazards (issue), their tolerable manifestations (measure), the possible external causes (adverse conditions, attacks, etc.) that can create an unreliable system, and the expected system reaction to mitigate the issues impact over the stakeholders. However, it does not discuss how to reconcile the stakeholders' value propositions on software quality/dependability attributes, how to avoid the one-size-fits-all metrics in defining quality attribute requirements, or how to define the appropriate levels of quality attribute requirements. It only provides stakeholders a template to present their quality requirements.

Furthermore, neither COM model nor UMD discusses how to use the metrics and model to drive the software quality achievement process. Nevertheless, both the COM model and the UMD can be complementary and integrated with our software analysis framework to form a set of "combined" frameworks for modeling software quality.

2.3 Process Strategies for Software Quality Achievement

The current software quality achievement process strategies can be categorized into two classes: fault avoidance and fault tolerance [Sommerville 2004]. Fault avoidance means that the system is developed in such a way that human error is avoided and thus system faults are minimized. Thus the development process is organized so that faults in the system are detected and repaired before delivery to the customer. For instance, verification and validation (V&V) techniques are used to discover and remove faults in a system before it is deployed. Fault tolerance means that the system is designed so that faults in the delivered software do not result in system failure.

Another categorization of software quality achievement process strategies is based on opportunity tree [Boehm 2001, Madachy-Lee 2003] as shown in Figure 2.3. It is compatible with the previous categorization. The software system quality is improved by reducing the defect risk exposure. The quality achievement process strategies can be categorized into two classes: 1) strategies to decrease defects (i.e., the probability of failure); 2) strategies to decrease defect impact (i.e., the size of loss due to a system failure). The first class can be further categorized into defect prevention strategies and defect detection and removal strategies. The second class can be further categorized to value/risk-based defect reduction strategies and graceful degradation strategies. Table 2.1 summarizes the software quality achievement process strategies based on the opportunity tree categorization.

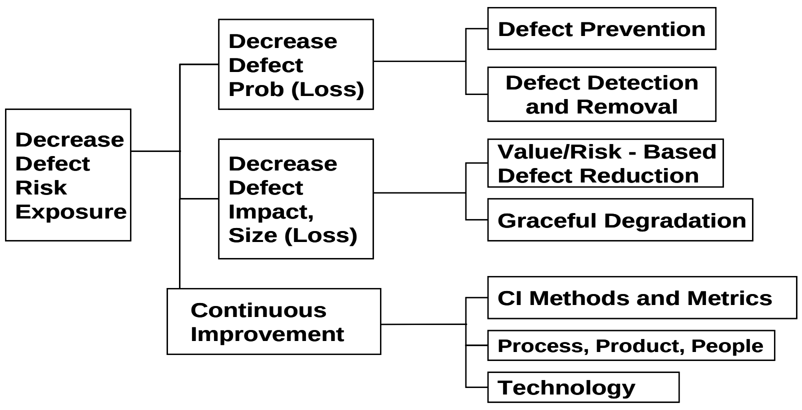

Decrease Defect Risk Exposure
Decrease Defect Prob (Loss)
Defect Prevention
Defect Detection and Removal
Decrease Defect Impact, Size (Loss)
Value/Risk - Based Defect Reduction
Graceful Degradation
Continuous Improvement
CI Methods and Metrics
Process, Product, People
Technology

Figure 2.3. Software Quality Achievement Opportunity Tree

Categories	Sub-categories	Process Strategies
Defect Prevention		Failure Modes & Effects Analysis [Leveson 1995], Fault Tree Analysis [Leveson 1995], Prototyping, User & Customer Involvement, Process automation, Reuse-oriented processes, Process maturity, Cleanroom process [Prowell 1999, Linger 1996]
Defect Detection and Removal	**Automated Analysis**	Completeness checking, Consistency Checking (views, interfaces, behavior, pre/post conditions), Traceability checking, Compliance checking (models, assertions, standards) Soft Goal approach [Chung 1999]
	Reviewing	Peer reviews, inspections Project Reviews Pair programming
	Testing	Requirements/design V&V, Inspections, Formal specification & verification Structural testing, Unit/Functional test Operational profile tests Usage (alpha, beta) tests Regression tests Test automation
Value/Risk-Based Defect Reduction		Value-based review [Lee 2005] Value/Risk-based testing [Gerrard 2002, Ramler et. al. 2006]
Graceful Degradation		Fault tolerance

Table 2.1. Software Quality Achievement Process Strategies Based on Opportunity Tree

Among those process strategies to improve software quality, the cleanroom software engineering process [Prowell 1999, Linger 1996], which emphasizes the defect prevention and avoidance, is relevant to part of our research objectives. It embodies a set of software engineering process principles as following:

- *Incremental development under statistical quality control*

  The cleanroom process employs the incremental development in which each increment is an iteration of the development process. It allows for continuous process improvement using the measurements (e.g., consecutive error-free random test cases, MTTF) taken during the incremental releases of the software which indicate whether the development process is in control or not. The process diagnosis and correction are necessary for the next iteration when the process is out of control.

- *Statistics-based software testing*

  The cleanroom process intends to validate the software requirements (including the system quality and reliability requirements) by constructing the system usage model which defines all possible uses of software functions and their probability distributions. And the testing environment is also statistically representative of the real operational environment. The testing and certification team can define the system usage in terms of Markov models which enables the automatic test case generation.

- *Mathematics-based software engineering for correct software designs*

  The cleanroom process treats the software program as a special case of a mathematical function. Since each function maps inputs onto outputs. The set of all possible inputs is treated as the function domain and the set of all correct outputs is the function range.

The University of Toronto’s Soft Goal approach [Chung 1999] presents a framework for representing and analyzing non-functional requirements (NFRs) using the software goal approach. It proposes a tool based on Non-Functional Requirements (NFRs) to achieve conflicting goals by decomposing the goals, analyzing design tradeoffs, rationalizing design decisions, and evaluating goal achievement. It also illustrates how the NFR framework helps the developer to deal with non-functional requirements and to drive the software development process rationally. Since it focuses on requirements traceability with more emphasis on incorporating changes in NFRs, it helps detect defects systematically and supports the process of corresponding changes in design and implementation rather than resolving conflicts among different stakeholders. Therefore, it does not consider stakeholders with different concerns and dependencies on quality attributes.

However, we believe that the sensible way of approaching software quality is to define its achievement from the perspectives of different stakeholders. Neither the cleanroom process nor the Software Goal approach considers the stakeholders' roles and their value propositions so that their metrics for the quality attributes of the software system tends to be one-size-fits-all.

Value-based review [Lee 2005] , risk-based testing [Gerrard 2002], and value-based testing [Ramler et. al. 2006] are process strategies that are the most compatible with our research objective. [Lee 2005] proposes a value-based review procedure and shows the payoff of value-based review checklist and procedure compared with the value-neutral ones. [Gerrard 2002] introduces the risk-based testing process strategy and its applications in E-business for achieving various quality objectives. [Ramler et. al. 2006] describes the practices supporting the management of value-based testing and outlines a framework for value-based test management. They focus on a specific stage of software development rather than the entire development life cycle.

2.4 Value-Based Software Engineering (VBSE) Research and Its Addressal of Software Quality Challenges

Determining whether software is *sufficiently correct* requires understanding both the level of software quality required for a particular application and the level of quality provided by the software. Further, the level of quality is multidimensional: for example, some applications depend on low latency but can tolerate low precision; in other applications precision is critical but latency is not. It follows that software that is acceptable in one situation may be deficient in another [Shaw 2002]. This view of software quality is rooted in the value proposition of engineering. Engineering seeks timely, cost-effective solutions to practical problems, preferentially solutions based on results that are well grounded in mathematics and science. This entails reconciling conflicting constraints and making design decisions with limited time, knowledge, and resources. This sets the objective of software engineering as creation of overall value, not simply the creation of functional and extra-functional capability. For instance, the user-centered design must consider the needs and preferences of users. This blurs the line between correctness and quality, because different users have different needs, their needs change over time, and they may be inarticulate about the differences between preferences that reflect dissatisfaction and preferences that reflect actual failure.

As different stakeholders depend on different system capabilities in different situations, software quality is necessarily a multi-attribute construct whose attribute values are situation-dependent. Based on the above observations, researchers have developed, with encouraging results, general frameworks for reasoning about various aspects of software quality. At the Economics-Driven Software Engineering Research Workshop [Sullivan et. al. 1999-2005] and elsewhere [Reifer 2002, Nejmeh 2002], researchers have developed general frameworks for making software engineering decisions about enhance the value of delivered software systems. In addition to our work, other EDSER contributions that explicitly address quality aspects include:

- Carnegie Mellon University's work on value-based security investment analysis [Butler 2002], warranty models for software [Li 2002], and value-based software fault detection [Raz 2001].
- The University of Virginia's application of real-options theory to the value of modularity [Sullivan 1999] and application of utility theory and stochastic control approaches to reliable delivery of computational service [Cai 2002].

For example, [Butler 2002] develops a technique for selecting an appropriate suite of security technologies for a particular computer installation. Different installations must protect different resources; they have different budgets and different concerns about security threats. Selection of security technology for an installation should begin with a quantitative analysis of that installation's risks, but the staff is usually not able to quantify either the frequency of possible attacks or the consequences of a successful attack. Butler adapts the techniques of multi-attribute decision theory to create a risk analysis technique that elicits subjective comparisons that the installation staff is able to make and convert these comparisons to quantitative figures of merit that can be used for subsequent analysis. Given this analysis, Butler considers the countermeasures available in the marketplace. Combining threat analysis with countermeasure information, she identifies candidate technologies, performs sensitivity analysis, and iterates the analysis with the security staff of the installation. The process leads both to recommendations for security technology acquisition and to deeper understanding of security issues on the part of the installation security staff.

However, none of the above provides a complete value-based framework to bridge the gap between stakeholder needs and the quality achievement for a software project.

In [Aurum et. al. 2005], Boehm presents an overview and agenda for Value-Based Software Engineering (VBSE) in Chapter 1. He discusses the seven key elements that provide candidate foundations for value-based software engineering with a case study in Chapter 6. In Chapter 2, he proposes an initial “4+1” theory of Value-Based Software Engineering (VBSE). The engine of the “4+1” theory is the stakeholder win-win Theory W, which addresses the questions of “which values are important?” and “how is success assured?” for a given software engineering enterprise. The four additional theories that it draws upon are utility theory (how important are the values?), decision theory (how do stakeholders’ values determine decisions?), dependency theory (how do dependencies affect value realization?), and control theory (how to adapt to change and control value realization?). After discussing the motivation and context for developing a VBSE theory and the criteria for a good theory, the chapter discusses a seven-step process-oriented expansion of the "4+1" VBSE theory framework for defining, developing, and evolving software-intensive systems. It also illustrates the application of the theory to a supply chain system example.

The Value-Based Software Quality Achievement (VBSQA) process proposed in the dissertation research is highly compatible with the general "4+1" VBSE theory, but focuses on reasoning about the software quality attributes and achieving quality goals using the stakeholder/value-based approach.

Other value-based view of software quality includes [Emam 2003], which investigates the Return On Investments (ROI) of software quality. It emphasizes the pre-release and post-release cost savings due to defect reduction in calculating the profit and ROI. However, it does not take into account the business value that a software system generates for its stakeholders at a certain quality level. And it does not discuss various stakeholders' dependencies on different quality attributes based on their value propositions in a software project. In addition, defect corrective cost only accounts for 21% of post-release software maintenance costs. Besides, 79% costs are due to requirement changes including 25% adaptive, 50% perfective, and 4% perfective changes [Lientz-Swanson 1978].

Chapter 3

Value-Based Software Quality Analysis Framework

The survey of the related literature in the area of software quality reveals that despite the development of various software quality attribute modeling techniques and their related analyses, the proper definitions and modeling of software quality attributes from the perspectives of stakeholders' value propositions appear to be lacking. However, it's critical to understand both the level of software quality required for a particular application by its success-critical stakeholders and that level provided by the software system. At the same time, a software quality achievement process driven by stakeholders' value propositions is also expected to achieve software quality in a cost-effective way. Based on these observations, we propose the stakeholder/value-based approach to leverage the software quality modeling and analysis. These also enable us to use the value-based software quality attribute definitions and models to drive a software development process to achieve stakeholder mutually satisfactory software quality requirements.

In this chapter, the research foundation of Value-Based Software Quality (VBSQ), a Value-Based Software Quality Analysis Framework, is described. The framework is composed of VBSQ attribute definitions, a Value-Based Software Quality Model (VBSQM), and a Value-Based Software Quality Achievement (VBSQA) process. Section 3.1 presents the definitions and metrics of VBSQ attributes. Section 3.2 describes the integrative VBSQM with its three components and the two usage scenarios of the VBSQM. Section 3.3 proposes the VBSQA process.

3.1 Software Quality As Stakeholder Value

3.1.1 Stakeholders' Views of Software Quality

In real world software development, there are many views of software quality so that there also exist many ways to suboptimize its achievement on the incomplete views. For instances, the freedom of personal health information in the health care system will compromise the privacy of patients because the exposure of such information may be abused by some agencies to check the patients' medical or credit history. However, overemphasis on privacy may also produce an unusable system. An example is the SAL conference room scheduling system built by a CS577 project team. The end users finally felt that the system is not so convenient to use by imposing overwhelming privacy protection on each reservation since sometimes they may want to know their neighboring reservations and discuss with each other to adjust the reservation schedule in person. The automated test case generator discussed in the Chapter 1 in [Aurum et. al. 2005] shows that overemphasis on completeness in testing wastes resources since the value of the test cases usually follows the Pareto 80-20 distribution.

Therefore we should reconsider the issue of software quality from a promising perspective that has been developed over the past five years: the value-based perspective. This perspective recognizes that technical decisions can often be made best when they are informed by explicit models of costs, risks, opportunities, and benefits. These models allow one to determine how an engineering decision will affect the ultimate value of a system, and thereby provide a rational approach to making tradeoffs.

In the meantime, complex missions or projects involve a large and heterogeneous group of stakeholders with various quality perspectives and different (sometimes even conflicting) needs. Ideally, one would like to have a single quality metric by which the development process could be driven, and by which the contributions of each technology could be ranked. However, in practice, such a one-size-fits-all metric is unachievable. Different systems have different success-critical stakeholders, and these stakeholders depend on the system in different ways. For example, the mean time between failures for a user depending on acceptable response time will be different from the MTBF for an operator just depending on system liveness (a real stakeholder value conflict on the Earth Observation System Distributed Information System (EOSDIS)). Therefore, “quality” might have different meanings for different software application domains, different software systems. And even for the same system, different stakeholders might have different views and definitions of its “quality”.

Thus, a critical first step in understanding the nature of software quality is to identify the success-critical stakeholder classes for a software system, and to characterize the relative strengths of their dependencies on various attributes of a given information system. This involves answering three main questions:

1. What are the primary quality attributes of a software system that success-critical stakeholders depend on?
2. What classes of stakeholders exhibit different patterns of dependency on these attributes?
3. For each class of stakeholder, what is the relative strength of their dependency on each attribute?

Based on the understanding of the stakeholders' dependency on quality attributes, again ideally, one would like to derive a set of attribute weights that could be combined into a single metric that could be used as the criterion for evaluating this (kind of) system's quality achievement. However, the strengths of such dependency may vary by operational context or mission scenarios so that the traditional value-neutral quality metrics and models become unsuitable.

Finally, the increasing pace of software system change requires more lightweight and adaptive processes, while the increasing mission-criticality of software systems requires more process predictability and control, as well as more explicit attention to business or mission values. Thus we aim to propose a software development process as a guide to help project stakeholders achieve their expected/desired levels of quality attributes using the value-based quality metrics, models and methods.

3.1.2 VBSQ Attribute Definitions

VBSQ attribute definitions differ from traditional value-neutral definitions in that they explicitly reflect the relevant success-critical stakeholders' value propositions. Traditional definitions such as Mean Time Between Failures (MTBF) for Reliability are referenced to the properties that stakeholders are relying on, such as liveness, accuracy, or performance. Our definitions go beyond traditional one-size-fits-all definitions of these attributes in reflecting their variability not only due to changes in operational profiles but also due to differences in stakeholder value dependencies. The definitions also try to bring some order to the definitions of such terms as safety, survivability, security, and privacy.

3.1.2.1 Protection Attributes: Safety, Security, Privacy

There are many different definitions of "safety", "security", and "privacy". We have tried to propose definitions that are value-based, minimally ambiguous, compatible with major current standards, and reasonably orthogonal. The resulting orthogonal definition framework is shown in Table 3.1.

Nature of Risks	**Causes of Risks**	
	Authorized Operations, Nature Causes	Unauthorized Operations
Physical Risks	Safety	Safety, Security
Information Risks	Privacy	Privacy, Security

Table 3.1. A Definition Framework for Safety, Security, and Privacy

<u>Current Definitions of Safety, Security, Privacy</u>

In terms of the nature of the risks being protected, we use the IEEE-1228 [IEEE 1994] standard on Software Safety Plans as a baseline. It defines "software safety" as "freedom from software hazards;" "hazard" as "a software condition that is a prerequisite to an accident;" and "accident" as "an unplanned event or series of events that results in death, injury, illness, environment damage, or damage to or loss of equipment or property."

Thus, the definition of “safety” focuses on the protection of physical entities (death, injury, illness, damage to equipment, property, or the environment). In contrast, the definition of “Information Security” in NIST Special Publication 800-37 [Ross-Swanson 2003] focuses on protection of information assets. Its definition is “The protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability.” Protection of “confidentiality” includes personal privacy and proprietary information. “Integrity” protects against improper information modification and destruction, and includes non-repudiation and authenticity. “Availability” protects timely and reliable access to information.

All of these protections are focused on the information-entity domain. However, the information security practices in NIST 800-37 also include the value-based activity of risk assessment. The standard metric for the criticality of an information security threat or vulnerability is its risk exposure *RE*, defined as *RE* = *P(Loss)* * *M(Loss)*, where *P(Loss)* is the probability of lost value and *M(Loss)* is “the magnitude of harm that a loss of confidentiality, integrity, or availability will have on an agency’s operations, assets, or individuals should the exploitation occur.” Thus, the scope of information security practice extends to the domain of physical entities as well.

The counterpart hazard analysis metric for software safety is *RE* = *P(hazard)* * *M(hazard)* [Leveson 1995]; it is the core metric for software risk management [Boehm 1989] and is also applied to privacy losses.

The risks involved in Privacy are generally limited to information-domain financial or personal reputation losses rather than losses in the physical domain. However, privacy protection for individuals also extends to constraints on violating privacy by authorized operations as well as unauthorized operations, as shown in Table 3.1.

Special Cases

There are some special cases discussed above that blur the orthogonality presented in Table 3.1, and some additional ones as well. These are summarized below.

- Information assets include both pure-information assets (data, programs) and physical assets (storage devices, processors).
- Financial assets are property, but are now almost exclusively electronic records in the information domain.
- The value of property loss may be measured financially within private institutions, but in public institutions must consider fairness to the least-advantaged stakeholders. Thus, a fire-dispatching algorithm to minimize the financial value of property loss that saves a few rich people’s houses while letting many poor people’s houses burn is not an acceptable solution for a public institution.
- The financial value of a human life is too controversial a concept to be merged with the financial value of property, and is generally considered separately in practice. Some partial exceptions are voluntary contracts such as insurance policies.

- Providing security protection from unauthorized operations must also consider protection from vulnerabilities caused by authorized operations or natural causes, such as accidents that provide open access to secure information.

Value-Based Definitions of Safety, Security and Privacy

In this context, our value-based definitions of safety, security, and privacy are as follows:

A system provides Safety to the extent that it minimizes stakeholders' expected loss of value due to death, injury, illness, or damage to equipment, property, or the environment.

A system provides Security to the extent that it minimizes stakeholders' expected loss of value from unauthorized access, use, disclosure, disruption, modification, or destruction of information assets, including financial losses and loss of value due to death, injury, illness, or damage to equipment, property, or the environment.

A system provides Privacy to the extent that it minimizes stakeholders' expected loss of value from authorized or unauthorized access, use, disclose, or modification of stakeholders' personal information, including financial losses and loss of reputation.

3.1.2.2 Robustness Attributes: Reliability, Availability, Survivability

Reliability:

A system provides Reliability to the extent that it maximizes the probability that the system will provide stakeholder-desired levels of service (liveness, accuracy, performance, others) with respect to a system's operational profile (probability distributions of transaction frequencies, task complexities, workload volumes, others) over a given period of time.

Discussion. This definition implies that the Reliability probability number will be different for the same system and operational profile (e.g., the percentage distribution of the adverse natural conditions, adversarial actions and normal conditions), if there are different stakeholders relying on the system for different desired levels of service. This may seem untidy for those wishing to have a single, value-neutral, one-size-fits-all Reliability number (e.g., with the definition of what is counted as failure referenced to a one-size-fits-all severity scale). But the different numerical outcomes add value by indicating to stakeholders that the system may be sufficiently reliable for some stakeholders but not for others. This equates to a stakeholder win-lose situation, which most often will turn into a lose-lose situation if the losing stakeholders lose cooperative motivation or try to manipulate the situation into a win for them and a losing situation for others. An example was provided at one point of the operation of the NASA Earth Observation System Distributed Information System (EOSDIS). The system operators were incentivized to optimize the system for liveness, which often left the users with 2-week response times to queries. An example lose-lose user response is to flood the system with whatever queries one might need results for at some point. In this case, managing to two different Reliability numbers, one for liveness probability and another for (say) 10-minute response time probability, would be better. And the alternative is to have a single value of Reliability that is irrelevant or misleading to most of the stakeholders.

The major practical implication of this situation is that any Reliability numbers furnished to stakeholders need an accompanying definition of the assumptions about operational profiles and stakeholder values used in deriving the numbers.

<u>Availability</u>:

A system provides <u>Availability</u> to the extent that it maximizes the fraction of time that the system will provide stakeholder-desired levels of service with respect to a system's operational profile.

<u>Discussion</u>. As with Reliability, a system's Availability number will vary across stakeholders with different desired levels of service (it also varies across different operational profiles). Where *MTBF* and Mean Time to Repair (*MTTR*) are relevant, Availability can be expressed as *MTBF/(MTBF + MTTR).*

The definition of Reliability and Availability assume a fixed value associated with the term "desired level of service" (such as full-service liveness or 1-second response time). From a value standpoint, their maximizing of the probability or fraction of time that the system provides this level of service is equivalent to minimizing the risk exposure *RE* due to losses in providing the desired level of service. The Availability version of *RE* is generally preferred, since it is sensitive to *MTTR* while the Reliability version is not. Since the values of Reliability and Availability are scenario-dependent (as may be the magnitude of loss), the overall *RE* is generally a probability-weighted sum of the scenarios' risk exposures:

$$RE = \sum_{S} P_S \cdot P(Loss)_S \cdot M(Loss)_S$$

(S – Scenario)

Survivability:

However, frequently the stakeholders will prefer a reduced level of service (such as core-capability liveness or 3-second response time) to having no service at all. An alternative way to maximize the system's expected value or minimize its risk exposure is to provide reduced-loss fallback capabilities when the full desired level of service is unachievable. This situation provides a definition of Survivability:

A system provides Survivability to the extent that it maximizes the total expected value obtained from achieving stakeholder-desired levels of service and from reduced levels of service when the desired levels of service are unachievable. The risk exposure *RE* associated with Survivability is defined in terms of the probabilities and magnitudes of reduced levels of service for each scenario, $P_{R/S}$ and M_{RS}:

$$\text{RE} = \sum_{S} P_S \cdot \sum_{RS} (P(Loss)_{R/S} \cdot M(Loss)_{RS})$$

(*S* – Scenario, *RS* – Reduced Levels of Service)

3.1.2.3 Quality of Service Attributes: Performance, Accuracy, Usability

Performance:

A system provides Performance to the extent that it maximizes the value of processed information achievable within the available resources (i.e., processors, storage devices, communication bandwidth, etc.) being used to process the system's workload (the volume and distribution of requested services/functions over a given time period). For information utilities in which value cannot be determined (e.g., Google), an alternate definition is that a system provides Performance to the extent that it provides stakeholders with their desired information with minimum utilization of limited resources and response time.

Discussion. Different stakeholders will have different values for different aspects of the system performance (e.g., response time, degree of accuracy) since they depend on the system in different ways. Furthermore, even the values desired by the same stakeholder can vary with the operational context or mission scenarios. Thus a one-size-fits-all metric for system performance is inadequate. In some cases in which different performance levels are differently priced, total system revenue can serve as a proxy for value of processed information.

Accuracy:

As with response time, accuracy is a value-neutral metric used as an acceptability level in such value-based attributes as Reliability and Performance. A system provides Accuracy to the extent that it minimizes the difference between delivered computational results and the real world quantities that they represent.

Discussion. There are several accuracy metrics, including RMS (Root Mean Square), absolute value, median or mode. Consistency is a sub-attribute of accuracy; a system's results can be useful for comparative analysis (tradeoffs, make vs. buy) by being consistent within some uniform bias, even if not being exactly accurate.

Usability:

A system provides Usability to the extent that it maximizes the value of a user community's ability to benefit from a system's capabilities with respect to the system's operational profile (probability distributions of transaction frequencies, task complexities, workload volumes, others). User community here refers to the information brokers, information consumers and system controllers in Table C.3 (see Appendix C).

Discussion. The measurement of Usability will be based on multiple factors: avoidance of misuse, accessibility, controllability, understandability, ease of learning, etc. A software system can be accessed in different ways by various stakeholders. For instance, the system administrator may access an interface different than what information consumers access. Therefore the measurement of Usability should vary across stakeholders based on their usage of the system. Additionally it should also be scenario-driven.

3.1.2.4 Other Attributes

Evolvability:

A system provides Evolvability to the extent that it maximizes the added value achievable in modifying the system or component in desired/valued directions within a given time period.

Discussion. Evolvability has several sub-attributes: structure, understandability and verifiability. Various stakeholders will have different criteria in assessing the evolvability of the software system since they depend on the system in different ways. Furthermore the measurement of evolvability is scenario-driven. In some scenarios, it becomes a special case such as portability.

Interoperability:

A system provides Interoperability to the extent that it maximizes the value of exchanging information or coordinating control across co-dependent systems.

Discussion. Interoperability has several sub-attributes: openness, understandability and verifiability. Openness means to be compliant with open interface standards. Similar to evolvability, interoperability is scenario-driven.

Correctness:

A system provides Correctness to the extent that its implementation precisely satisfies its requirements and/or design specifications.

Discussion. Correctness is another value-neutral attribute used as an acceptability level in such value-based attributes as Safety and Security. Various stakeholders have different concerns about correctness. It is a direct concern for developers or acquirers. However, acquirers will be more concerned about the correctness of mission-critical requirements. Users may prefer a slightly incorrect implementation of the requirements if it is easier to use.

Timeliness (Schedule):

A system provides Timeliness to the extent that it maximizes the value added by developing new capabilities within a given delivery time. On the other hand, if the set of desired capabilities is fixed, an alternate definition is that a system provides Timeliness to the extent that it minimizes the calendar time required to deliver the set of capabilities.

Discussion. This definition of Timeliness applies to both software/system development and maintenance. For maintenance, Timeliness is highly correlated with Evolvability. One approach to achieving Timeliness is the Schedule As Independent Variable (SAIV) process. It involves stakeholders in prioritizing their desired features, architecting for ease of adding or dropping borderline features, and incremental development of highest-priority features [Boehm, et. al. 2002].

Affordability (Cost):

A system provides Affordability to the extent that it maximizes the value added by developing new capabilities within a given budget.

Discussion. This definition of Affordability applies to both software/system development and maintenance. For maintenance, Affordability is also highly correlated with Evolvability. One approach to achieving Affordability is the Cost As Independent Variable (CAIV) process. It involves stakeholders in prioritizing their desired features, architecting for ease of adding or dropping borderline features, and incremental development of highest-priority features [Boehm et. al. 2002]. On the other hand, if the set of desired capabilities is fixed, an alternate definition is that a system provides Affordability to the extent that it minimizes the cost required to deliver the set of capabilities.

Reusability:

A system provides Reusability to the extent that it maximizes the Return On Investment (ROI) of reusing system capabilities in other products.

Discussion. Reusability is correlated with interoperability in that it depends on several sub-properties such as openness and simplicity of interfaces. It frequently works best within a domain-engineered product line architecture.

3.1.3 Value-Based Software Quality Metrics

Since different software systems have different stakeholders and there exist various scenarios in a software system, the traditional one-size-fits-all software quality metric is not applicable. Based on the value-based definitions of software quality attributes described in the previous section, the value-based metrics are proposed to measure software quality achievement. The key elements of value-based software quality metrics are summarized as follows:

1. Reflecting the success-critical stakeholders S_i related to the project
2. Reflecting the software quality attributes Q_j that success-critical stakeholders are depending on
3. Related operational scenarios to each quality attribute Q_j
4. A matrix as shown in Table 3.2 to track the stakeholder acceptable level AL_j, desired level DL_j, stakeholder dependency P_{ij} of quality attribute Q_{ij} and the actual progress A_j in realizing the level of quality attribute Q_j

Quality Attributes	**Related Scenarios**	**Acceptable Levels**	**Desired Levels**	**Stakeholders Dependencies**						**Actual Levels**
				S_1	S_2	…	S_i	…	S_n	
Q_1		AL_1	DL_1	P_{11}	P_{21}	…	P_{i1}	…	P_{n1}	A_1
Q_2		AL_2	DL_2	P_{12}	P_{22}	…	P_{i2}	…	P_{n2}	A_2
⋮		⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Q_j		AL_i	DL_i	P_{1i}	P_{2i}	…	P_{ii}	…	P_{ni}	A_j
⋮		⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Q_n		AL_n	DL_n	P_{1n}	P_{2n}	…	P_{in}	…	P_{nn}	A_n

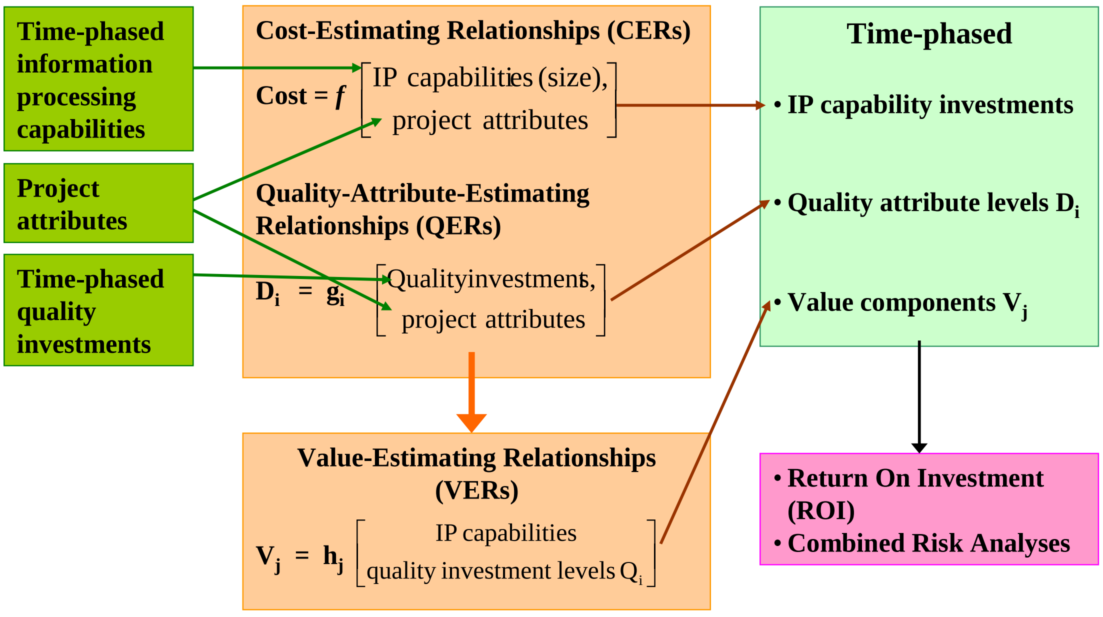
Table 3.2. Value-Based Metrics for Software Quality Attribute

3.2 Value-Based Software Quality Model (VBSQM)

Different stakeholders depend on different software quality attributes (e.g., availability, safety, security or performance) in different ways under different situations. Thus the business case for software quality must deal with multiple situation-dependent attribute values. However, in most organizations, proposed investments in software quality compete for limited resources with proposed investments in software and system functionality, speed of development, and other system capabilities. The lack of good return-on-investment (ROI) models for software quality causes difficulties for decision-makers in determining the overall business case for software quality investments, in determining which software quality investments are most cost-effective, and in determining how much software quality investment is enough. As a result, investments in software quality and the resulting system quality are frequently inadequate. Thus, software quality models will need to support stakeholders in determining their acceptable and/or desired levels for each quality attribute and estimating the cost, value, ROI and risk exposures for achieving those. On the other hand, competitions among software quality attributes also exist. Therefore, determining how much software quality investment is enough will also need to support tradeoff analyses among different software quality attributes. Along these lines, we develop a integrated Value-Based Software Quality Model (VBSQM) for reasoning about software quality's ROI and performing combined risk analyses using the COCOMO II [Boehm et. al. 2000a], COQUALMO [Steece et. al. 2002] models, and value-based approach. Section 3.2.1 introduces three components of the VBSQM. Section 3.2.2 proposes the two usage scenarios of the integrated VBSQM.

3.2.1 Components of VBSQM

The VBSQM integrates the cost estimating relationships (CER's) from the Constructive Cost Model COCOMO II; the software quality attribute estimating relationships (QER's) from the Constructive Quality Model COQUALMO; and the value estimating relationships (VER's) supplied by the system's stakeholders. The overall structure of the VBSQM is shown in Figure 3.1.



Time-phased information processing capabilities
Project attributes
Time-phased quality investments
Cost-Estimating Relationships (CERs)
Cost = f [IP capabilities (size), project attributes]
Quality-Attribute-Estimating Relationships (QERs)
D_i = g_i [Quality investments, project attributes]
Value-Estimating Relationships (VERs)
V_j = h_j [IP capabilities, quality investment levels Q_i]
Time-phased
• IP capability investments
• Quality attribute levels D_i
• Value components V_j
• Return On Investment (ROI)
• Combined Risk Analyses

Figure 3.1. The Overall Structure of Value-Based Software Quality Model (VBSQM)

In this section, we first summarize the empirical results on the relative cost of achieving increasing levels of some attributes of software quality. Section 3.2.1.1 presents results from the calibration of the Constructive Cost Model (COCOMO) II [Boehm et. al. 2000a] to 161 representative project data points. Section 3.2.1.2 shows related results from a COCOMO II extension called COQUALMO. It is based on fewer project data points, but is calibrated to data on software defect introduction and removal, showing the effects of investments in software defect detection via automated analysis, peer review, and execution testing on delivered defect density. Section 3.2.1.3 elaborates the relations between COCOMO II and COQUALMO which we use as a bridge to construct the VBSQM. Section 3.2.1.4 then presents example forms of value/utility functions relating value to achieved levels of software quality attributes for various classes of stakeholders.

3.2.1.1 Software Cost Model: COCOMO II

An initial set of cost estimating relationships (CER's) is provided by the COCOMO II model. The COCOMO II CER's enable users to express time-phased information processing capabilities in terms of equivalent software size, and to estimate time-phased software life cycle investment costs in terms of software size and the project's product, platform, people, and project attributes. Additional future CER's would include CER's for COTS-related software costs, inventory-based CER's for hardware components and COTS licenses, and activity–based CER's for associated investments in training and business process re-engineering.

The core of the Constructive Cost Model (COCOMO) II is a mathematical relationship involving 24 variables used to estimate the amount of effort in person-months required to develop a software product defined by the variables. By multiplying the project effort by its cost per person-month, one can also estimate the project's cost.

COCOMO II's parameters include the product's equivalent size in thousands of lines of code (KSLOC) or a function-point equivalent; personnel characteristics such as capability, experience, and continuity; project characteristics such as execution-time and storage constraints; and product characteristics such as complexity, reusability, and required reliability.

The effect of each variable on project effort has been determined by a Bayesian combination of expert judgement and multiple regression analysis of the data from 161 completed projects representing a wide range of sizes and applications. The regression analysis determined the size and significance of each parameter's effect on project effort. The effect of the "Required Software Reliability" (RELY) variable on project effort is shown in Figure 3.2.

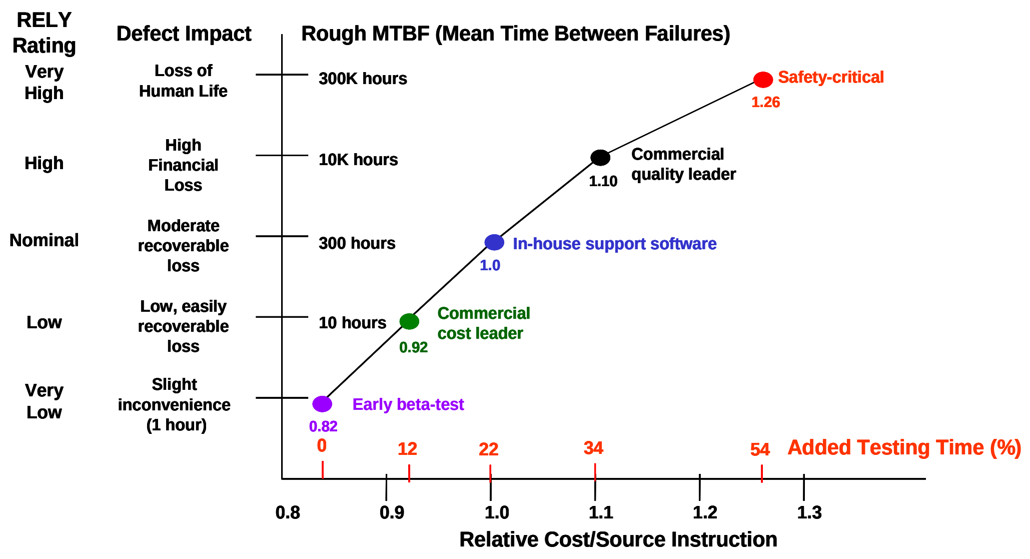


Figure 3.2. Software Development Cost vs. Required Reliability (RELY) Trade-off

The RELY rating scale is also shown at the left of Figure 3.2, in terms of the defect risk or impact of a defect on the product's operational behavior and outcome. The added effort for a Very High RELY project is the net result of rework savings due to early error elimination and extra effort in very thorough off-nominal, model-based, stress, and regression testing at the end of the project. Since the extra effort occurs near the end, when the project is about at its average staffing level, it roughly translates into an extra 54% of calendar time in thorough testing before fielding the product. The regression analysis of the 161 projects produced a relative effort range of 1.54 between projects reporting their required reliability (RELY) as Very Low (the impact of a product failure was a slight inconvenience) and projects reporting a Very High RELY rating (the impact of a product failure was a risk of loss of human life in a safety-critical system). The t-value produced by the regression analysis for the RELY variable was 2.602, well above the statistical significance level of 1.96 for this sample size and number of variables [Boehm et. al. 2000a; page 169].

The corresponding effort multiplier relative to a Nominal value of 1.0 shows the relative cost per source for each rating level, assuming that the rating levels of the other variables stay constant. Thus, for example, the relative cost of a safety-critical product will be 26% higher than a nominal in-house software product. This value represents the net effect of the added effort to prevent, detect, and fix more software defects versus the reduced rework effort resulting from earlier defect detection.

The above results are summarized in Figure 3.2. Based on data from a subset of the projects, we have also added a rough scale of product Mean Time Between Failures (MTBF) corresponding to the relative impact of product failures, going from 1 hour MTBF for Very Low RELY to 300K hours MTBF for Very High RELY, which are also shown in Figure 3.2. For instance, the low, easily recoverable losses associated with a Low RELY rating correspond to an MTBF of 10 hours, or roughly one serious failure per day; while a high RELY rating corresponds to an MTBF of 10, 000 hours, or about 1.14 years.

3.2.1.2 Software Quality Model: COQUALMO

An initial set of software quality attribute estimating relationships (QER's) is provided by the COQUALMO model. As an extension of the COCOMO model, COQUALMO enables users to specify time-phased levels of investment in improving dependability attributes, and to estimate the resulting time-phased dependability attribute levels. The current version of COQUALMO estimates delivered defect density in terms of a defect introduction model estimating the rates at which software requirements, design, and code defects are introduced, and a subsequent defect removal model. The overall structure of the COQUALMO model is shown in Figure 3.3.

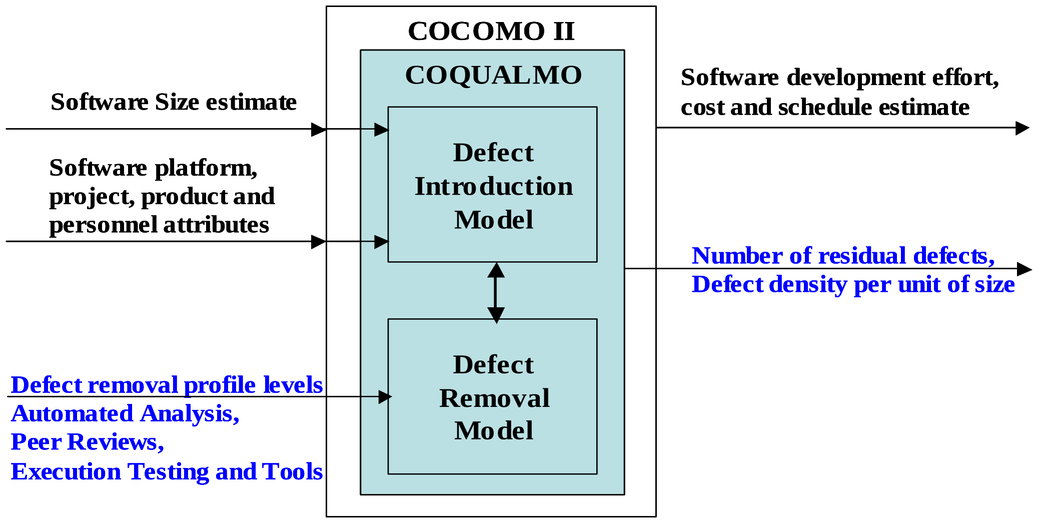

COCOMO II
COQUALMO
Software Size estimate
Software platform, project, product and personnel attributes
Defect removal profile levels
Automated Analysis,
Peer Reviews,
Execution Testing and Tools
Defect Introduction Model
Defect Removal Model
Software development effort, cost and schedule estimate
Number of residual defects,
Defect density per unit of size

Figure 3.3. The Overall Structure of COQUALMO Model

The defect introduction rates are determined as a function of calibrated baseline rates modified by multipliers determined from the project's COCOMO II product, platform, people, and project attribute ratings. For example, a Very Low rating for Applications Experience will lead to a significant increase in requirements defects introduced, and a smaller increase in code defects introduced. The defect removal model estimates the rates of defect removal as a function of the project's levels of investment in automated analysis tools, peer reviews, and execution testing and tools. Its rating scales are shown in Table 3.3 ranging from Very Low to Extra High.

The calibrated baseline (i.e., nominal) defect introduction rates for COQUALMO are 9 requirement defects/KSLOC, 19 design defects/KSLOC, and 33 code defects/KSLOC. For simplicity and to avoid unwarranted precision, we have rounded these to 10, 20 and 30, for a total of 60 defects/KSLOC introduced [Boehm et. al. 2000a]. Starting from this baseline, the COQUALMO estimation of reduced delivered defect density as a function of the composite defect removal rating is shown in Figure 3.4. The defect removal model estimates the rates of defect removal as a function of the project's levels of investment in automated analysis tools, peer reviews, and execution testing and tools. Its rating scales are shown in Table 3.3. Initial CER's are available to estimate the costs of these investments. The Very Low composite defect removal rating leaves delivered defect density to 60 Delivered Defects/KSLOC (DDK), while an Extra High rating can reduce the delivered defect density at only 1.6 DDK [Boehm et. al. 2000a; page 266]. The RELY Cost-Estimating Relationship (CER) in COCOMO II discussed in section 3.2.1.1 is available to estimate the costs of these investments, as its Very Low to Very High rating levels correspond to the horizontal rows of defect reduction investments in Table 3.3. Note that the composite defect removal rating is an integration of the ratings for automated analysis tools, peer reviews, and execution testing and tools. Note also that it assumes nominal rates of defect introduction: a strong defect prevention program can reduce delivered defect densities by another factor of 60 to 100. The RELY Cost-Estimating Relationship (CER) in COCOMO II discussed in section 3.2.1.1 is available to estimate the cost of these investments, as its Very Low to Very High rating levels correspond to the horizontal rows of defect reduction investments in Table 3.3. For mixed levels of investment in analysis, reviews, and testing, COQUALMO DDK estimates and an equivalent RELY rating can also be determined.

The current COQUALMO defect introduction model is calibrated to the total number of defects introduced, including bad fixes. This is a reasonable first approximation, but is insensitive to the defect removal rate. An example extension will be a more precise treatment of “bad fixes”, which average about 7% of all defect fixes and over 10% of defect fixing effort.

Further COQUALMO extensions will refine its current QER’s, and will provide further QER’s for estimation of additional dependability attributes such as performance and security [Reifer 2003].

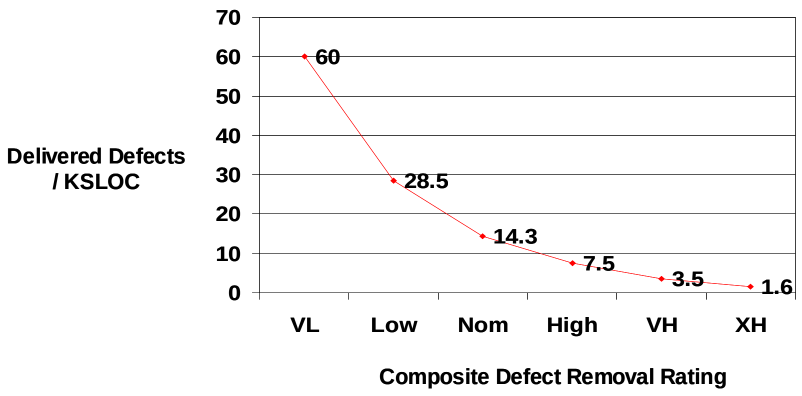


Figure 3.4. COQUALMO Reduced Delivered Defects Estimates at Nominal Defect Introduction Rates

Rating	Automated Analysis	Peer Reviews	Execution Testing and Tools
Very Low	Simple compiler syntax checking.	No peer review.	No testing.
Low	Basic compiler capabilities	Ad-hoc informal walkthroughs	Ad-hoc testing and debugging.
Nominal	Compiler extension Basic requirements and design consistency	Well-defined sequence of preparation, review, minimal follow-up.	Basic test, test data management, problem tracking support. Test criteria based on checklists.
High	Intermediate-level module and inter-module; Simple requirements/design	Formal review roles with well-trained participants and using basic checklists, follow up.	Well-defined test sequence tailored to organization. Basic test coverage tools, test support system. Basic test process management.
Very High	More elaborate requirements/design Basic distributed-processing and temporal analysis, model checking, symbolic execution.	Basic review checklists, root cause analysis. Formal follow-up using historical data on inspection rate, preparation rate, fault density.	More advanced test tools, test data preparation, basic test oracle support, distributed monitoring and analysis, assertion checking. Metrics-based test process management.
Extra High	Formalized specification and verification. Advanced distributed processing	Formal review roles and procedures. Extensive review checklists, root cause analysis. Continuous review process improvement.. Statistical Process Control.	Highly advanced tools for test oracles, distributed monitoring and analysis, assertion checking Integration of automated analysis and test tools. Model-based test process management.

Table 3.3. COQUALMO Defect Removal Investment Rating Scales

3.2.1.3 Relations Between COCOMO II and COQUALMO

For reliability, the COCOMO II Required Reliability (RELY) CER provides an initial bridge to software quality estimation, expressed in terms of the operational impact of software defects (see Figure 3.2). Participants at a USC industry government Affiliates' Workshop translated this bridge into a rough experience-based QER for software reliability in terms of Mean Time Between Failure (MTBF) in hours (see Figure 3.2). Figure 3.2 also shows the corresponding effort multipliers (relative levels of effort or cost) to achieve the associated reliability levels, as calibrated from experience data on 161 diverse software projects. For example, developing software for users with low, easily recoverable losses (such as PC users) results in an MTBF of 10 hours (roughly a daily crash) and a relative cost of 0.92. Developing software for financial organizations, where crashes can cause high financial losses, results in an MTBF of 10K hours (417 days or somewhat over a year between crashes) and a relative cost of 1.10.

The relationship between COCOMO II and COQUALMO is based on the fact that the COQUALMO rating scales for levels of investment in defect removal via automated analysis, peer reviews, and execution testing and tools have been aligned with the COCOMO II RELY rating levels shown in Figure 3.2. The correspondence between COCOMO II RELY ratings and COQUALMO defect removal profile ratings is based upon a mapping between the activity analysis behind the COCOMO RELY effort multiplier (see Appendix A) and the COQUALMO defect removal activity ratings (see Appendix B). One can thus compare the levels of investment for the Low and High COCOMO II rating levels with the tools and activities assumed to be used at these levels in the COQUALMO rating scales. To cover the COQUALMO Extra High rating level in Table 3.3, we have provisionally extended the reliability rating scale in Figure 3.2 to Extra High, with a corresponding MTBF of 1M hrs and a relative cost of 1.56. This is based on some experiences with thorough independent verification and validation efforts, which added about 30 percent to software costs. For ratings between Very Low and Extra High, VBSQM provides two ways to apply this relationship between COCOMO II and COQUALMO. One is to specify a COCOMO II RELY rating and assume the same investment levels in automated analysis, peer reviews, and execution testing will be applied, in which case the corresponding relative effort and MTBF will be used. Or, we can specify our own investment levels for automated analysis, peer reviews, and execution testing and tools. Based on the specified effort distribution among the three categories of defect removal techniques, we can calculate a COQUALMO-based weighted average of these levels as the equivalent COCOMO II RELY rating. We could also use other cost models such as Knowledge Plan, PRICE S, SEER, and SLIM in place of COCOMO II, to the extent that they have a similarly defined RELY CER. The relationship between COCOMO II and COQUALMO also produces a way to relate investments in software reliability to resulting values of the delivered system's Mean Time Between Failures (MTBF), as shown in Figure 3.2.

3.2.1.4 Software Stakeholder Value Models

This section presents some example forms of utility functions relating the value of achieving levels of software quality attributes for various classes of stakeholders. Section 3.2.1.4.1 provides two functional forms relating software system availability (an attribute of Robustness) to stakeholder value. Section 3.2.1.4.2 elaborates and extends on the work in [Snir 2003] showing functional forms relating achieved information response time (an attribute of Performance) to stakeholder value. In the e-service domain, the major Value Estimating Relationships (VERs) involve losses in market share due to insufficient software quality and/or delayed product delivery. Section 3.2.1.4.3 presents value/untility functions supplied by project critical stakeholders to relate software quality levels or product delivery time to resulting benefit flows and value earned.

3.2.1.4.1 Value Estimating Relationships (VERs): Availability

The VBSQM needs initial software quality VERs supplied by the system's stakeholders, to relate estimated cost investments and software quality levels to resulting benefit flows and ROI estimates. VBSQM VERs assume that stakeholders have performed a baseline business-case analysis for various components of value (profit, customer satisfaction, on-time performance) as a function of the time-phased information-processing capabilities at baseline software quality attribute levels. VBSQM aggregates these value components into an overall time-phased value stream, comprising the time-phased costs (the cost of IP capabilities plus software quality investments) and normalized using present-value formulas to produce a time-phased ROI profile.

A typical value-versus-availability relationship can appear as a production function as shown in Figure 3.5. Below a certain level of investment, as with the gains in availability don't avoid bankruptcy. Beyond this level, there is a *high-returns segment*, but at some point, incremental gains in availability don't affect users' frustration levels, resulting in a *diminishing-returns segment*. The initial VBSQM VERs involve simple relationships such as the operational cost savings per delivered defect avoided, or the loss in sales per percent of the system downtime, shown as the linear approximation of a particular segment of production function in Figure 3.5. Many organizations providing e-services also use such relationships to measure loss of revenue due to system downtime. For example, on the higher side, Intel estimates its loss of revenue as $275K ($US) for every hour of order-processing-system downtime; other companies estimate $167K (Cisco), $83K (Dell), $27K (Amazon), $8K (E*Trade), and $3K (Ebay).

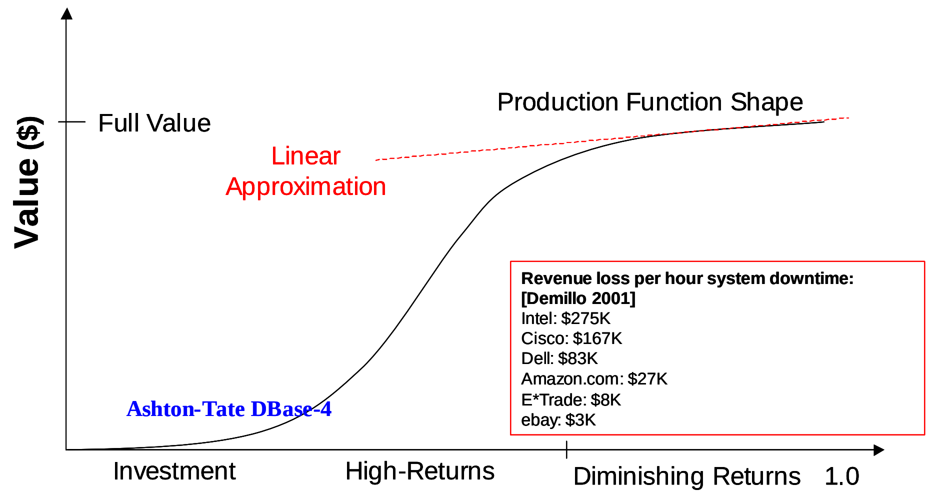


Figure 3.5. Typical Value Estimating Relationships (VERs) of Availability

3.2.1.4.2 Value Estimating Relationships (VERs): Operational Response Time

Stakeholder utility functions relating the value of receiving processed information to operational response time usually reflect the stakeholder value added by being able to make time-critical commitments based on better knowledge of the state of nature. A classic example was the use by the Rothschild family in London to obtain rapid information about the outcome of the Battle of Waterloo in 1815. When they were the first to learn of England's victory, they made a double killing by initially selling their shareholdings and causing shareholdings to plunge in value as others interpreted their action to reflect knowledge of an English loss. They then had time to quietly buy up the lower-price shares before the news generally arrived of England's victory.

The general shape of their utility function is shown in Figure 3.6. This shape of utility function is generally characteristic of information brokers, mission-critical users, and mission controllers in Table C.3. Having early and exclusive information about the state of nature enables them act in advance to achieve gains or avoid losses. Other examples are automobile drivers, military commanders, or urban fire-equipment dispatchers. Early information enables them to make decisions and deploy resources to maximum effect. As the window of opportunity closes, there is less time to exploit the information and the value goes down, becoming relatively zero when the window of opportunity to exploit the information closes.

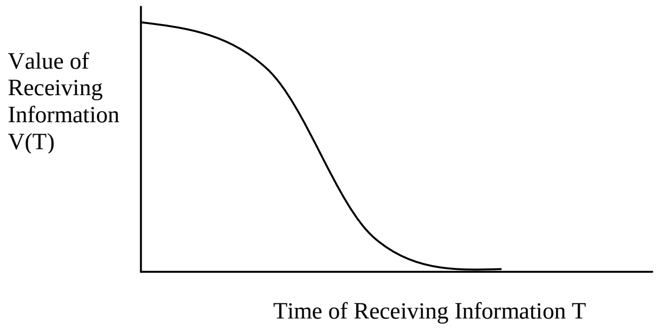


Figure 3.6. Value of Information Timeliness for Information Brokers, Mission-Critical Users, and Mission Controllers

The shape of the curve in Figure 3.6 will vary by stakeholder and by situation. For example, a farmer or storekeeper would not be able to benefit much from advance information about the Battle of Waterloo and would have a flatter utility function, while a newspaper publisher would realize different benefits than the Rothschilds from near-term sale of newspapers and longer-term gains in reputation.

For some real-time mission controllers such as in spacecraft operations or financial closing deadlines, the utility function V(T) becomes a step function as in Figure 3.7. Here, having the information anytime before the deadline is equally valuable, but having it after the deadline has no value.

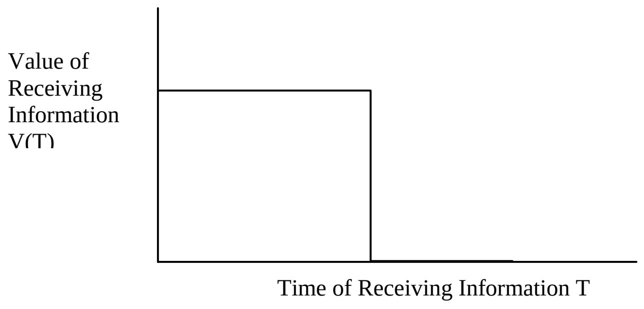


Figure 3.7. V(T) for Real-Time Mission Controllers

For some non-critical information users, the value of information vs. time may be relatively flat. For example, astronomical information or historical archives will retain their value fairly uniformly as a function of time, as in Figure 3.8.

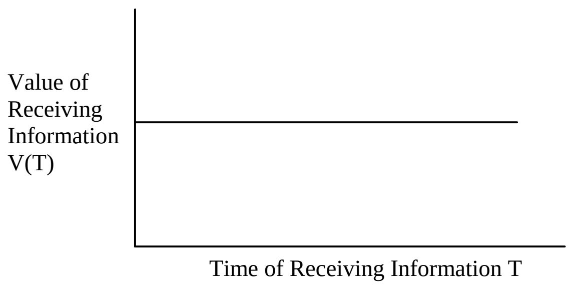


Figure 3.8. V(T) for Non-Critical Information Users

Some information system administrators use step-function value structures to enable users to buy prioritized information processing or information access. Thus, for example, users wishing near-instant information access will pay a higher rate, while users satisfied with one-hour, four-hour, or overnight response will pay lower rates, as in Figure 3.9.

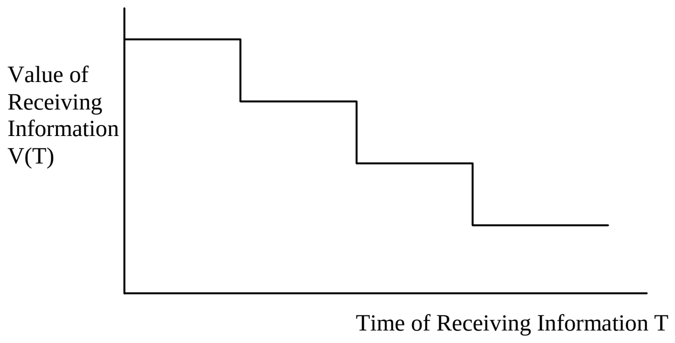


Figure 3.9. V(T) for Some Administrators

3.2.1.4.3 Value Estimating Relationships (VERs): Value Loss vs. System Delivery Time

Since schedule is one of the software quality attributes, it is also a project critical success factor to determine how much software quality investment is enough. Thus project stakeholders have to be involved in determining the value estimating relationships (VERs) for the system delivery time in different project business cases. The initial VERs for system delivery time show different types of stakeholder value/utility functions for relating the mission/market value loss vs. time of delivery. They usually reflect the "cost of delay" in missed opportunities to make time-critical commitments due to the delayed delivery of a software system. In this section, we describe three types of value/utility functions for system delivery time. In this section, we describe three types of value/utility functions for system delivery time based on different project business cases.

The usual shape of the value/utility function for the case of marketplace competition is the classic S-shaped economic production function shown in Figure 3.10. This shape of utility function is generally characteristic of software projects such as e-services and wireless networking infrastructure. Early delivery of the system enables them to rapidly capture market share ahead of their competitors. As the time of delivery passes a specific point, the market share diminishes and the system value loss goes up rapidly until reaching a diminishing-returns point, when there is very little market share left to lose.

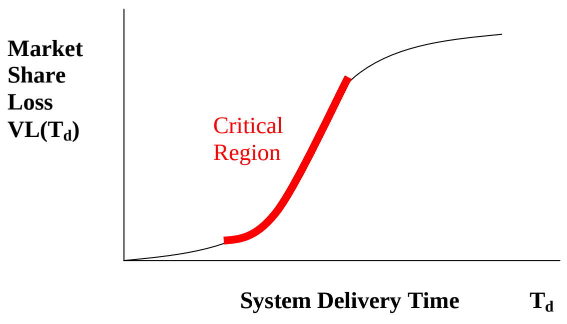


Figure 3.10. Marketplace Competition (Internet Services, Wireless Infrastructure): Value Loss vs. System Delivery Time

The shape of the value/utility function for the case of fixed-schedule event support is shown in Figure 3.11. For software systems to support some fixed-schedule events such as the Olympic games, a trade show or a Mars Rover launch window, the value function $VL(T_d)$ becomes a step function. Here, system delivered before the deadline loses no value, but missing the deadline loses all the value.

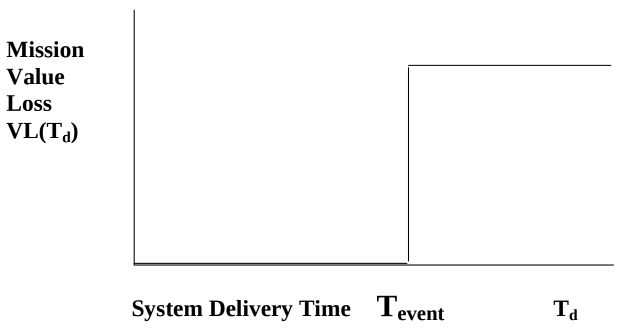


Figure 3.11. Fixed-schedule Event Support: Value of On-time System Delivery

For off-line data processing systems, the user value loss vs. system delivery time may be relatively flat. For example, scientific applications for astronomical information or historical archives will retain their value fairly uniformly as a function of delivery time, as shown in Figure 3.12.

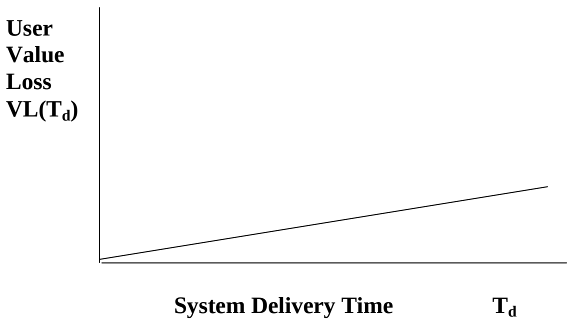


Figure 3.12. Off-line Data Processing: Value Loss vs. System Delivery Time

3.2.2 Integrating Framework: Value-Based Software Quality Model (VBSQM)

The VBSQM integrating the COCOMO II, COQUALMO and VERs provides two usage scenarios to support software quality analysis from the stakeholder/value perspectives. It helps to answer such questions as how much software quality investment is enough in terms of both ROI and combined risk exposures. Section 3.2.2.1 presents the procedure of VBSQM software quality ROI analysis. Section 3.2.2.2 describes the procedure of VBSQM combined risk analyses of both software quality and market share erosion.

3.2.2.1 VBSQM Usage Scenario 1: Software Quality ROI Analysis

The integrated VBSQM framework can help project stakeholders and/or decision-makers to quantitatively determine an appropriate software quality level for a particular software project, project scenario class or software class. Such software quality ROI analysis is based on the cost and generated value of software quality investments. The ROI is computed as ROI = (Value – Cost)/Cost. The usage scenario of the VBSQM ROI analysis is as following:

1. Use a checklist of software quality attributes to involve stakeholders in prioritizing attributes of highest concern and usage scenarios.
2. Estimate software size in terms of value-adding capabilities.
3. Enter the size and baseline cost drivers into COCOMO II to obtain baseline cost estimates.
4. Enter baseline and alternative software quality drivers into COCOMO II and COQUALMO and obtain alternative cost and quality estimates.
5. Involve stakeholders in determining the appropriate form and parameters for value estimation relationships.
6. Apply VBSQM to assess the costs, benefits, and ROI’s for the alternatives.
7. Iterate previous steps as appropriate.

3.2.2.2 VBSQM Usage Scenario 2: Combined Risk Analyses

A classical problem facing many software projects is how to determine when to stop testing and release the product for use. We have found that risk analysis helps to address such "how much is enough?" questions, by balancing the risk exposure (probability of loss times size of loss) of doing too little with the risk exposure of doing too much. However, people often find it difficult to quantify the relative probabilities and sizes of loss in order to provide practical approaches for determining a risk-balanced "sweet spot" operating point.

Under the assumptions discussed in sections 3.2.1.1 through 3.2.1.3, the framework of VBSQM, which integrates the empirically-calibrated COCOMO II and COQUALMO results and quantified stakeholder-supplied VERs such as those discussed in section 3.2.1.4 , also provides the basis for us to perform combined risk analyses in order to solve the problem of how much software assurance is enough [Huang-Boehm 2005a].

The usage scenario of VBSQM for combined risk analyses on both software quality assurance and market share erosion is as following:

1. Estimate software size in terms of value-adding capabilities.
2. Enter the project size and cost drivers into VBSQM to obtain project delivered defect density (= (defects introduced – defects removed)/KSLOC) for the range of "Required Reliability" driver (RELY) ratings from Very Low to Very High.
3. Involve stakeholders in determining the sizes of loss $S_q(L)$ based on the value estimating relationships for software quality attributes.

4. Involve stakeholders in determining the risk exposures of market erosion RE_m based on the delivery time of the product.
5. Apply the VBSQM to assess the probability of losses for the range of "Required Reliability" cost driver (RELY) ratings from Very Low to Very High based on the relative delivered defect density.
6. Apply the VBSQM to combine the software quality risk exposure and market erosion risk exposure to find the sweet spot.

We have extended the initial VBSQM discussed in [Boehm-Huang 2004a] to support such combined risk analyses. It provides the default values of size of loss due to unacceptable software quality $S_q(L)$ and risk exposure of market erosion RE_m for each RELY rating under three business cases (i.e., early start-up, normal commercial and high finance). Users can also provide their own values for $S_q(L)$ and RE_m based on their project business case. After the user inputs the project size in SLOC and rates each COCOMO II cost driver except RELY according to their own project situation, VBSQM will automatically generate the curve for combined risk exposure and help to locate the sweet spot for their software quality investment level.

3.3 Value-Based Software Quality Achievement (VBSQA) Process

3.3.1 Purpose of the VBSQA Process

Another key objective of this research is to use the value-based software quality definitions, metrics and models to drive the software development process in order to achieve the stakeholder mutually satisfactory software quality outcome. Ideally, one would like to have a single software quality metric by which the development process could be driven, and by which the contributions of each technology could be ranked. However, in practice, different systems have different success-critical stakeholders, and these stakeholders depend on the system in different ways. Furthermore, stakeholders often have emerging, time-varying software quality requirements. Therefore, using traditional one-size-fits-all software quality metrics and models to drive the system and software development process is likely to lead to delivered systems that are unsatisfactory to some stakeholders. Along these lines, we propose a Value-Based Software Quality Achievement (VBSQA) Process generated from the WinWin Spiral Model’s risk-driven approach. It is coupled with a set of value-based software quality analysis methods and models for reasoning about software and system quality. It helps project success-critical stakeholders to define, negotiate and develop mission-specific combinations of software quality attributes for the development of a system with the stakeholder WinWin-balanced software quality outcome.

3.3.2 Top-level Steps of the VBSQA Process Framework

This section presents the major steps, activities and decision points of the Value-Based Software Quality Achievement (VBSQA) process framework [Huang 2005] and identifies the techniques that can be applied in each step. Table 3.4 summarizes the major steps of VBSQA process. Each process step is to be elaborated in the following subsections as needed.

1.	Identify top-level mission objectives and stages – including quality (Q-) objectives
2.	Perform project cost/benefit analysis – Estimate project budget – Develop results chain to identify success-critical stakeholders and their top-level value propositions
3.	Stakeholders negotiate mutually satisfactory (Win-Win) quality (and other) goals and relevant mission scenarios.
4.	Risk-based process strategy decision making
5.	Concurrently engineer top-level Q-attribute and other requirements and solution tradeoff spaces.
6.	Identify top-level Q-risks, execute risk-mitigation spirals.
7.	Develop system top-level design and initial Feasibility Rationale Description (FRD).
8.	**Hold Life Cycle Objective (LCO) Review** – **Pass: go to 9. Fail: go to 5.**
9.	Concurrently engineer detailed Q-attribute and other requirements and solutions; resolve risks.
10.	Develop system detailed design and detailed Feasibility Rationale Description (FRD).
11.	**Hold Life Cycle Architecture (LCA) Review** – **Pass: go to 12. Fail: go to 9.**
12.	Construct, test, and deploy system – Use the mission scenarios and Q-attribute requirement levels as progress metrics and test cases – **Core Capability Demo (CCD)** – Monitor progress and change requests; perform corrective actions
13.	**Initial Operational Capability (IOC) Readiness Review**

Table 3.4. The Top-level Steps of VBSQA Process Framework

3.3.2.1 Identify Top-level Mission Objectives and Stages

This is the entry to the VBSDA process. Top-level software quality objectives are identified with top-level mission objectives.

3.3.2.2 Perform Project Cost/Benefit Analysis

Besides the project budget estimation, we have to perform stakeholder/value dependency analysis [Boehm-Huang 2004b] in order to understand the nature of the software quality. That is, we need to identify the major classes of success-critical project stakeholders, and to characterize the relative strengths of their dependencies on various attributes in each operational scenario of the software system. The generalized top-level stakeholder/value dependencies for information-intensive software systems as shown in Table C.3 (See Appendix C) can be a start point to identify the project success-critical stakeholders. At this step, a software quality-elaborated Results Chain shall be constructed to help identify the success critical quality-oriented initiatives and stakeholders. The Results Chain technique, developed by the DMR Consulting Group [Thorp 1998] is a way to identify missing initiatives and success-critical stakeholders in a system development project. It involves initially defining the project's Initiatives (rectangles), Contributions (arrows), Outcomes (circles, ovals), and Assumptions (hexagons) for its nominal-case operation. It then involves identifying risks and vulnerabilities that may go wrong with the nominal case, and establishing additional Initiatives, Contributions, and Outcomes to avoid or resolve them.

3.3.2.3 Stakeholders Negotiate Software Quality Goals and Relevant Mission Scenarios

Project success-critical stakeholders are involved in the WinWin negotiation in order to converge on the mutually satisfactory software quality (and other) goals. At this step, a stakeholder/goal matrix should be constructed based on the identified mission objectives/goals in step 1. This is also the starting/ending point of each spiral in the WinWin Spiral Model. Table 3.5 shows a sample matrix of the primary success-critical stakeholders in an information-intensive software system as rows and their prioritized goals with respect to the software system development, operation, and evolution as columns. The specific columns represent the primary categories of system goals/requirements to be negotiated by the stakeholders. Project goals and requirements include desired constraints on the system and project such as choices of programming language, infrastructure packages, and computing platforms; development and operational standards; and constraints on budgets, schedules, and other scarce resources. Capability goals include the functions that the software system should perform. Interface goals include message formatting and content, and interaction protocols with other interoperating systems. Level of Service goals include the dependability attributes, except for cost and schedule (covered under Project goals) and interoperability (covered under Interface goals). Evolution goals include downstream goals that the initial system architecture should support, such as deferred capabilities or scalability to accommodate workload growth. At the same time, each stakeholder class shall prioritize the goals as High, Medium and Low.

Goals / Stakeholders	Project	Capability	Interface	Level of Service	Evolution
Information Suppliers					
System Dependents					
Information Brokers					
Information Consumers					
System Controllers					
Developers					
Maintainers					
System Administrators					
System Acquirers					

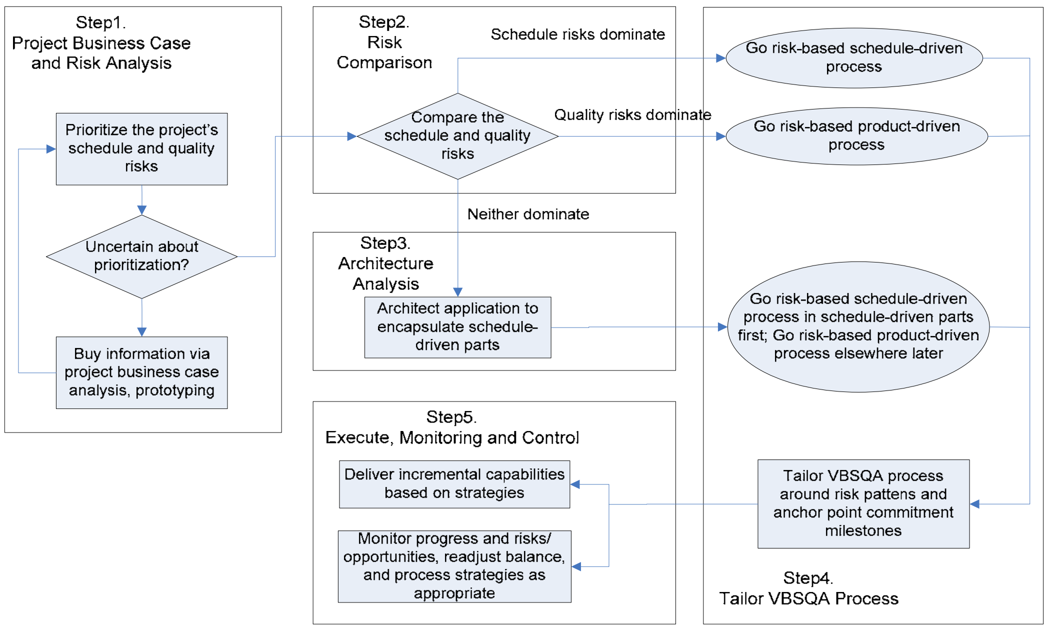
Table 3.5. A Sample Stakeholder/Goal Matrix

Instead of using one number to define each software quality goal, the value-based quantitative software quality models/tools such as VBSQM can be used to determine an appropriate level for some particular attributes (e.g., Reliability, Availability, etc.) based on the Return On Investment (ROI) profile and/or the tradeoffs between quality attributes and delivery time of the project. For instance, the VBSQM can help stakeholders determine how much availability is enough for different types of projects, different operational scenarios or different software classes in a certain project. The principles and the usage scenarios of the VBSQM are elaborated in section 3.2.

3.3.2.4 Risk-Based Process Strategy Decision Making

VBSQA process framework covers all the phases and milestones in the entire software development life cycle of the WinWin Spiral model [Boehm-Hansenzz, 2001]. It also includes various software development activities to incorporate the value-based consideration. In real-world software projects, different software quality assessment criteria are set based on different business cases [Reifer 2002] so that different process strategies should be selected to meet them. Therefore, a flexible process generation platform is required to enable the trim or addition of the steps/milestones/activities in the VBSQA process framework.

Along these lines, the risk-based process decision-making approach [Huang et. al. 2006a], summarized in Figure 3.13, uses the project business case and risk analysis to tailor the VBSQA process into an overall software development strategy [Boehm-Turner 2004]. This approach relies heavily on project key stakeholder identification, project business case analysis and the collaboration of the core development team and the project stakeholders. Thus this process decision making approach is embedded into the current VBSQA process framework following Step 3 (stakeholders negotiate software quality and other goals) shown in Table 3.4. Three process strategies (schedule-driven, product-driven and market-trend driven) can be selectively applied in the software development based on different project business cases. In general, schedule-driven processes are lightweight processes that employ short iterative cycles while product-driven processes employ longer iterative cycles.



Step1.
Project Business Case and Risk Analysis
Prioritize the project's schedule and quality risks
Uncertain about prioritization?
Buy information via project business case analysis, prototyping
Step2.
Risk Comparison
Schedule risks dominate
Compare the schedule and quality risks
Quality risks dominate
Neither dominate
Go risk-based schedule-driven process
Go risk-based product-driven process
Step3.
Architecture Analysis
Architect application to encapsulate schedule-driven parts
Go risk-based schedule-driven process in schedule-driven parts first; Go risk-based product-driven process elsewhere later
Step5.
Execute, Monitoring and Control
Deliver incremental capabilities based on strategies
Monitor progress and risks/ opportunities, readjust balance, and process strategies as appropriate
Tailor VBSQA process around risk pattens and anchor point commitment milestones
Step4.
Tailor VBSQA Process

Figure 3.13. Risk-based Process Decision-making Approach

Project business case analysis is used to elicit success-critical stakeholders' value propositions. Risk analysis is used to define and address risks particularly associated with project schedule and software quality achievement. It is also used to answer such question as "How much software quality investment is enough?" by balancing the risk of investing *too little* on software quality with the risk of investing *too much*. Examples of such questions related to software quality achievement are "How much prototyping is enough?", "How much review is enough?", and "How much testing is enough?". As another aspect of quality achievement, we extend the approach to also consider the question "How much architecting and planning is enough?". Risk analysis is closely related to business case analysis in that project risks are prioritized based on the business case analysis by emphasizing the high-priority stakeholder value.

If schedule risks dominate quality risks, risk-based schedule-driven process is applied. If quality-risks dominate schedule risks, risk-based product-driven process is applied. If neither dominates, then architect the application to encapsulate the schedule-driven parts which applies the risk-based schedule-driven process and go risk-based product-driven process elsewhere. Based on this approach, we can tailor the VBSQA process framework and establish an overall project strategy by integrating individual risk mitigation plans [Boehm-Turner 2004].

Since no decision is perfect for all time, as indicated in Step 5 in Figure 3.13 the project management team needs to continuously monitoring and controlling the performance of the selected process in order to adapt to changes in the business case. In this way, we can always monitor and control the opportunity for realizing stakeholders' value.

3.3.2.5 Concurrently Engineer Top-level Q-attribute and Other Requirements and Solution Tradeoff Spaces; Identify Top-level Q-risks, Execute Risk-mitigation Spirals

Step 5 and 6 are often coupled with each other during the software development process. The stakeholder/value dependency analysis and concurrency are two important factors in the VBSQA process. Simple deterministic processes are inadequate to address the emergent, time-varying priorities for dependability attributes. The VBSQA process generated from the WinWin Spiral Model provides a workable framework for dealing with risk-driven concurrency. On the other hand, it is effective in avoiding one-size-fits-all metrics and resolving the value conflicts and in software quality achievement.

In this section, a scenario-based approach is proposed to identify stakeholders' value propositions on software quality (Q-) attributes and help stakeholders define the detailed Q-attribute requirements for different scenarios. This approach also helps identify and resolve value conflicts on Q-attributes and to perform tradeoff analysis on Q-attributes in order to engineer stakeholder WinWin-balanced software Q-attribute requirements. Figure 3.14 shows the process elements for stakeholders to engineer top-level software Q-attribute requirements, identify software Q-risks and select the most cost-effective architecture/technology combination to mitigate risks for different scenarios. The entry criteria of the Q-attribute requirement engineering and risk mitigation are shown in the box in the top-left corner of Figure 3.14. Each process element is elaborated as needed.

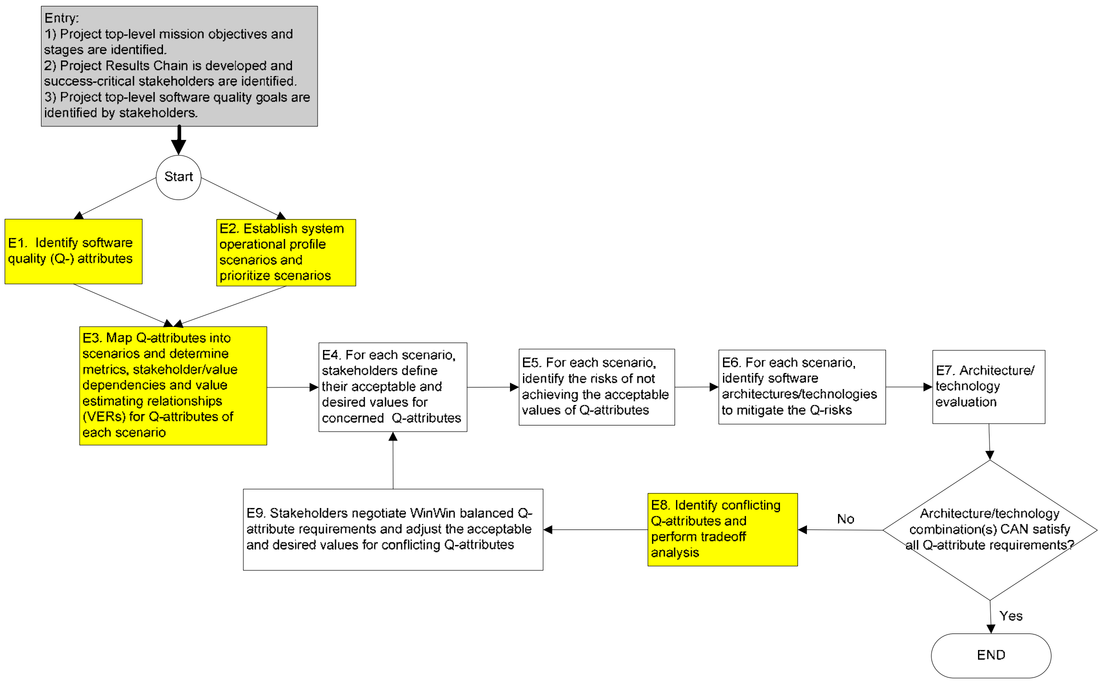

Entry:
1) Project top-level mission objectives and stages are identified.
2) Project Results Chain is developed and success-critical stakeholders are identified.
3) Project top-level software quality goals are identified by stakeholders.
Start
E1. Identify software quality (Q-) attributes
E2. Establish system operational profile scenarios and prioritize scenarios
E3. Map Q-attributes into scenarios and determine metrics, stakeholder/value dependencies and value estimating relationships (VERs) for Q-attributes of each scenario
E4. For each scenario, stakeholders define their acceptable and desired values for concerned Q-attributes
E5. For each scenario, identify the risks of not achieving the acceptable values of Q-attributes
E6. For each scenario, identify software architectures/technologies to mitigate the Q-risks
E7. Architecture/ technology evaluation
Architecture/technology combination(s) CAN satisfy all Q-attribute requirements?
No
Yes
END
E8. Identify conflicting Q-attributes and perform tradeoff analysis
E9. Stakeholders negotiate WinWin balanced Q-attribute requirements and adjust the acceptable and desired values for conflicting Q-attributes

Figure 3.14. A Scenario-Based Approach to Engineer Software Q-attribute Requirements and Risk Mitigation Plans

E1. Identify software quality (Q-) attributes

This is the entry of the software Q-attribute requirement engineering and risk mitigation process activities, where the top-level Q-attributes for the entire software project are established. The results obtained from Step 1 and 3 in the VBSQA process are usually used as the inputs of this step.

E2. Establish system operational profile scenarios and prioritize scenarios

The scenario can be defined as mission sequences, environmental inputs or Q-objective threats and their frequencies. A scenario is used to describe a proposed use case of the system and/or an interaction of one of the stakeholders with the system [Clements 2002]. Scenarios provide a vehicle for converting vague Q-attribute requirements into concrete use cases of a system and make Q-attribute requirements measurable and testable. The top-level scenarios of a software system can be established from its use case description (e.g., MBASE Operational Concept [MBASE 2003] use case description). A complex scenario can be decomposed into several component scenarios if it's necessary for testing purposes. On the other hand, several component scenarios can be composed into a high-level scenario for analysis or testing purposes. We provide a framework with three factors to be associated with each scenario S_i, which will be directly leveraged in our scenario-based approach:

- ***Value (v).*** The value loss (can be measured either in dollars or in utility) if a scenario execution fails. It indicates the impact of a scenario on the total mission value.
- ***Probability of occurrence (p).*** The probability that a scenario occurs in a specific mission mode. When several scenarios have the comparable value

impact on the entire mission, a scenario that is more frequently executed affects the software system quality more extensively, than if it were less. In a given mission mode, $\sum p_i = 1$. The operational profile of a mission mode can then be established based on the scenario probability distribution.

- ***Software Q-attribute metrics (m).*** All scenarios are mapped into Q-attributes based on their relevance. For instance, if we intend to measure the reliability of a scenario, we may use *MTBF* as a metric. Scenarios can then be prioritized based on their value (v) and probability of occurrence (p).

In addition, we can use a matrix as shown in Table 3.6 to track the effect on improving a Q-attribute (e.g., Reliability, Availability) on the project based on the scenario distribution if the Q-attribute covers several scenarios or the entire project. Using Reliability (measured by *MTBF*) as an example, in Table 3.6, p_i represents the probability of the occurrence of scenario i, v_i represents the associated value of scenario i, $MTBF_i$ represents the *MTBF* of scenario i, and Σ represents the Combined *MTBF* of all the scenarios for a mission mode based on the scenario distribution defined by p_i.

Scenario i	***1***	***2***	***…***	***i***	***…***	***n***	***Σ***
p_i							-
v_i							-
Before applying an architecture/technology strategy							
MTBF$_i$							-
Failures/max(*MTBF**$_i$)***							-
Exp. Fail's/max(*MTBF**$_i$)***							
After applying an architecture/technology strategy							
MTBF$_i$							-
Failures/max(*MTBF**$_i$)***							-
Exp. Fail's/max(*MTBF**$_i$)***							

Table 3.6. A Sample Matrix Tracking the Improvement of a Q-attribute Based on Scenario Distribution

- In general, we can compute the Combined *MTBF* before and after applying an architecture/technology strategy using the following formula:

$$Combined\ MTBF = \frac{\max(MTBF_i)}{\sum p_i \dfrac{\max(MTBF_i)}{MTBF_i}}$$

- Availability can be obtained if the $MTTR_i$ is also known.

<u>E3. Map Q-attributes into scenarios and determine metrics, stakeholder/value dependencies and value estimating relationships (VERs) for Q-attributes of each scenario</u>

Software Q-attributes are mapped into each scenario based on their relevance. The metric for a Q-attribute may be different in different scenarios. For instance, Performance can be measured in response time (s) or in storage space (MBytes) in different scenarios.

The stakeholder/value dependency analysis can be performed for each scenario based on the stakeholders' top-level value propositions obtained in Step 2 (see Table 3.4). That is, we need to identify the major classes of success-critical project stakeholders, and to characterize the relative strengths of their dependencies on various attributes of each scenario of the software system.

If needed, the value estimating relationships (VERs) of each Q-attribute can be also established based on the impact of the Q-attribute on a particular scenario. Note that the VERs for a Q-attribute may also be different in different scenarios since the same Q-attribute's impact on different scenarios may be different.

E4. For each scenario, stakeholders define their acceptable and desired values for concerned Q-attributes

The results of the VBSQM ROI analysis to determine the appropriate levels of certain Q-attributes (e.g. Availability, Reliability) for different scenario classes discussed in section 3.3.2.3 can be used as guidance for stakeholders to define their expected and desired levels for these Q-attributes.

Similarly, stakeholders can define their expected and desired levels for other Q-attributes based on the priority of a particular scenario and their value dependencies on the scenario-related Q-attributes.

E5. For each scenario, identify the risks of not achieving the acceptable values of Q-attributes

E6. For each scenario, identify software architectures/technologies to mitigate the Q-risks

E7. Architecture/technology evaluation

The top-level risks associated with each operational scenario should also be identified at this stage. Scenario-based Fault Tree Analysis (FTA) [Leveson 1995], Failure Modes and Effects Analysis [Leveson 1995] and Dependability Cases [Weinstock 2004] are three useful techniques to trace scenario failures to the potential risks causing them. Note that FTA is primary a means for analyzing causes of failures, not identifying failures. The top event in the tree must have been predicted and identified by other technique such as scenario-based approach discussed in this section. Such FTA or Dependability Case can be started from analyzing the sequence diagrams of each scenario. The root of the Fault Tree is a particular system usage scenario. The leaves of the Fault Tree are potential risks of not achieving the expected values of Q-attributes pertain to the scenario.

Risks are quantitatively linked to the Q-attributes of each scenario. For each pair of risk and Q-attribute in each scenario, stakeholders provide an estimate (expert judgement) of the potential impact of the risk in the scenario. We define the "impact" as the proportion of the scenario value that would be lost were that risk occur. The probability of occurrence of each risk is also estimated.

As the start of executing risk-mitigation spirals, at least one architecture/technology strategy should be identified to address each risk item. For instance, we may use the attribute/strategy tables in MBASE Feasibility Rationale Description (FRD) [MBASE 2003] to find the existing strategies. As for the new architecture/technology, the architecture/technology developers should formulate their hypotheses in terms of the defect classes that can be avoided/detected by their architecture/technology and estimate the cost of applying the architecture/technology in order to prepare for the architecture/technology evaluation. Architecture/technology strategies are quantitatively linked to Q-risks. For each pair of Q-risk and architecture/technology strategy in each scenario, we provide an estimate (expert judgement) of the mitigation of the risk in the scenario. We define the "mitigation" as the proportion by which the risk would be reduced were that architecture/technology strategy to be applied. At the same time, the cost/effort of applying a particular architecture/technology strategy should also be recorded.

Tools such as the JPL risk-centric Defect Detection and Prevention (DDP) model [Feather 2002] can be used for such analysis.

E8. Identify conflicting Q-attributes and perform tradeoff analysis

E9. Stakeholders negotiate WinWin balanced Q-attribute requirements and adjust the acceptable and desired values for conflicting Q-attributes

If the existing architecture/technology strategies can't satisfy the acceptable values of all the Q-attributes, or if the estimated cost/schedule to satisfy all the Q-attribute requirements is too high, then the tradeoff function between the conflicting Q-attributes will need to be constructed and the tradeoff analysis will be performed in conjunction with additional stakeholder negotiation. An initial Q-attribute tradeoff model based on the calibrated COCOMO II model can help us perform

Cost/Schedule/RELY tradeoff analyses [Boehm-Huang 2004b, Huang-Boehm 2005b].

Multi-attribute preference analyses [Keeney 1993] and stakeholder win-win negotiation support tools [WinWin 2006] are useful techniques to help stakeholders perform such negotiations based on their value propositions. Based on the Q-attribute tradeoff analysis results, we can selectively relax the lower priority D-attribute requirements for the low risk scenarios using SCQAIV (Schedule/Cost/Quality As Independent Variable) process principles [Boehm et. al. 2002].

Note that such conflicting Q-attribute tradeoff analysis can be performed concurrently with the architecture/technology evaluation.

3.3.2.6 Develop System Top-level Design and Initial Feasibility Rationale Description (FRD)

Top-level design of at least one architecture option should be provided by developers. And the initial Feasibility Rationale Description (FRD) [MBASE 2003] furnishes the rationale for the product being able to satisfy the stakeholders' system requirements and specifications including the Q-attribute requirements. The initial FRD in LCO stage also includes an initial business case analysis (i.e., cost, benefits and ROI analysis) based on the Results Chain.

The initial risk analysis should identify all the major risks and propose an initial risk mitigation plan. Risks without mitigation in the LCO stage have to be resolved in the Life Cycle Architecture (LCA) stage.

3.3.2.7 Hold Life Cycle Objective (LCO) Review

An LCO Review is to be held with the participation of all the project key stakeholders. This indicates a milestone of the LCO phase in the WinWin Spiral Model. The exit criteria of LCO ARB are to provide at least one feasible architecture to satisfy the requirements, and to provide proofs of requirement satisfaction including the Q-attribute requirements. The result of the LCO review is either *Pass* or *Fail*:

- If *Pass*: go to step 9 (see Table 3.4) . Key stakeholders commit to support the project to proceed to its Life Cycle Architecture (LCA) phase.
- If *Fail*: go to step 5 (see Table 3.4)

3.3.2.8 Concurrently Engineer Detailed Q-attribute and Other Requirements and Solutions; Resolve risks

It follows the similar process discussed in section 3.3.2.5 but in a more detailed level.

3.3.2.9 Develop System Detailed Design and Detailed Feasibility Rationale Description (FRD)

System detailed design needs to be developed for only one feasible architecture. And the LCA Feasibility Rationale Description (FRD) [MBASE 2003] has to provide the detailed rationale of all requirement satisfaction including the Q-attribute requirements. The LCA FRD risk analysis should propose a detailed risk mitigation plan to resolve all known risks.

3.3.2.10 Hold Life Cycle Architecture (LCA) Review

An LCA Review is to be held with the participation of all the project key stakeholders. This indicates a milestone of the LCA phase in the WinWin Spiral Model. The LCA Feasibility Rationale Description (FRD) [MBASE 2003] has to provide the detailed rationale of all requirement satisfaction including the Q-attribute requirements. LCA FRD risk analysis should propose a detailed risk mitigation plan to resolve all known risks. The exit criteria of Life Cycle Architecture (LCA) Review is to commit one architecture to satisfy all the requirements of the system. The result of the LCO review is either *Pass* or *Fail*:

- If *Pass*: go to step 12 (see Table 3.4). Key stakeholders commit to support the full life cycle of the project and the project can proceed to the construction phase.
- If *Fail*: go to step 9 (see Table 3.4).

3.3.2.11 Construct, Test, and Deploy System

During the construction phase, we still need to monitor the progress of the software Q-attribute achievement and perform corrective actions when needed. We can use the mission operational scenarios and Q-attribute requirement levels defined by project key stakeholders as progress metrics and test cases.

The framework of the value-realization feedback process [Boehm-Huang 2003] shown in Figure 3.15 can be applied to monitor the progress of the software quality achievement and change requests. The software quality-elaborated results chain, business case, and risk mitigation plans set the baseline in terms of expected time-phased software quality investments, benefit flows, return on investment (ROI), project risk exposures, and underlying assumptions. As the project performs to plans, the actual or projected achievement of software quality investments, benefit flows and the assumptions' realism may become invalid, at which point the project team will need to determine and apply corrective actions by changing plans or initiatives, making associated changes in expected cost and benefit flows.

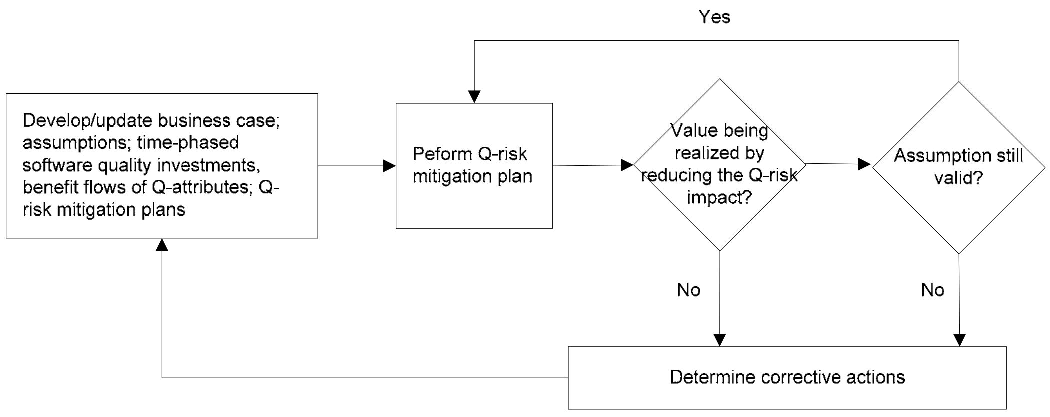

Yes
Develop/update business case; assumptions; time-phased software quality investments, benefit flows of Q-attributes; Q-risk mitigation plans
Peform Q-risk mitigation plan
Value being realized by reducing the Q-risk impact?
Assumption still valid?
No
No
Determine corrective actions

Figure 3.15. A Value-Realization Feedback Process to Monitor and Control the Achievement of Software Q-attribute Requirements

A matrix with the capability to track the value-based expected versus actual outcomes (e.g., software quality investments, reduced value loss, ROI) [Boehm-Huang 2003] is a useful technique to support the monitoring and control of the actual progress of the software quality achievement. Such matrix and the expected benefits and business case analyses work together to provide a means of tracking actual progress in realizing the benefits and applying corrective action wherever

- the expected benefits are not being realized,
- the expected cost and/or schedule are overrun,
- the assumptions in the results chain are becoming invalid, or

- new opportunities may surface with a higher payoff than the program being executed

Core Capability Demo (CCD) can be added as an intermediate milestone to improve the stakeholders' confidence in the software system delivery. Project success-critical stakeholders are invited to participate in the CCD to provide feedback on the developed system core capabilities.

3.3.2.12 Initial Operational Capability (IOC) Readiness Review

This is the final milestone of the software development life cycle. The Release Readiness Review (RRR) is held with the participation of all the success-critical stakeholders. Developers are required to deliver the final product with all the documents. If the RRR is passed, developers will perform the "cold turkey" transition of the software system to other stakeholders. If it fails, developers may be required to adjust or fix the product based on RRR feedback. Otherwise, the project will be announced as a failure.

3.3.3 Mapping VBSDA Process into "4+1" VBSE Theory

To show the compatibility of VBSQA Process with the "4+1" Value-Based Software Engineering (VBSE) Theory presented in Chapter 2 of [Aurum et. al. 2005], we here map the activities of VBSQA Process into the "4+1" VBSE Theory. The purpose of this work is to show a sound theoretical support of the VBSDA Process.

Figure 3.16 summarizes the overall structure of the "4+1" Theory of VBSE [Aurum et. al. 2005]. The engine in the center is the success-critical stakeholder (SCS) win-win Theory W. The core of Theory W is the Enterprise Success Theorem, that is, your enterprise will succeed if and only if it makes winners of your success-critical stakeholders. However, the Enterprise Success Theorem does not tell us how to achieve and maintain a win-win state which requires the involvement of the four supporting theorems to address the following four aspects:

1. Identifying all of the success-critical stakeholders (SCSs). (Dependency Theory)
2. Understanding how the SCSs want to win. (Utility Theory)
3. Having the SCSs negotiate a win-win set of product and process plans. (Decision Theory)
4. Controlling progress toward SCS win-win realization, including adaptation to change. (Control Theory).

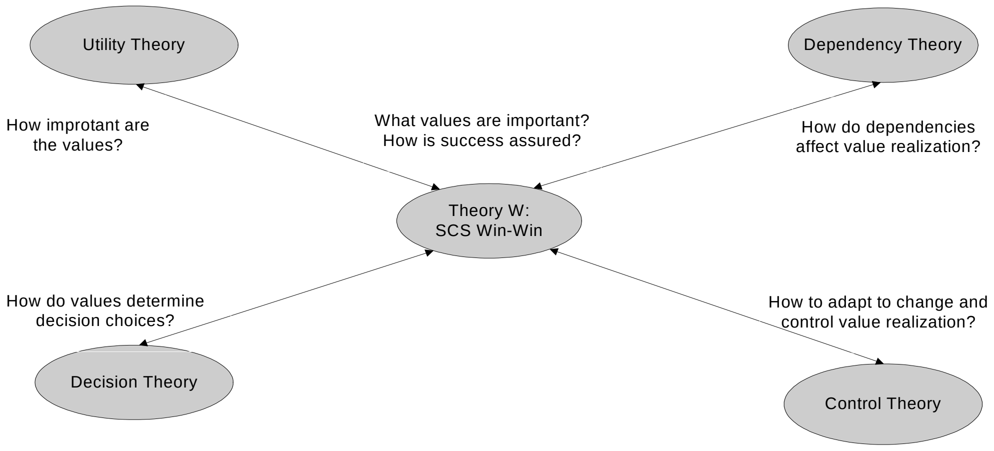

Utility Theory
Dependency Theory
How improtant are the values?
What values are important? How is success assured?
How do dependencies affect value realization?
Theory W: SCS Win-Win
How do values determine decision choices?
How to adapt to change and control value realization?
Decision Theory
Control Theory

Figure 3.16. The "4+1" Theory of VBSE: Overall Structure [Aurum et. al. 2005]

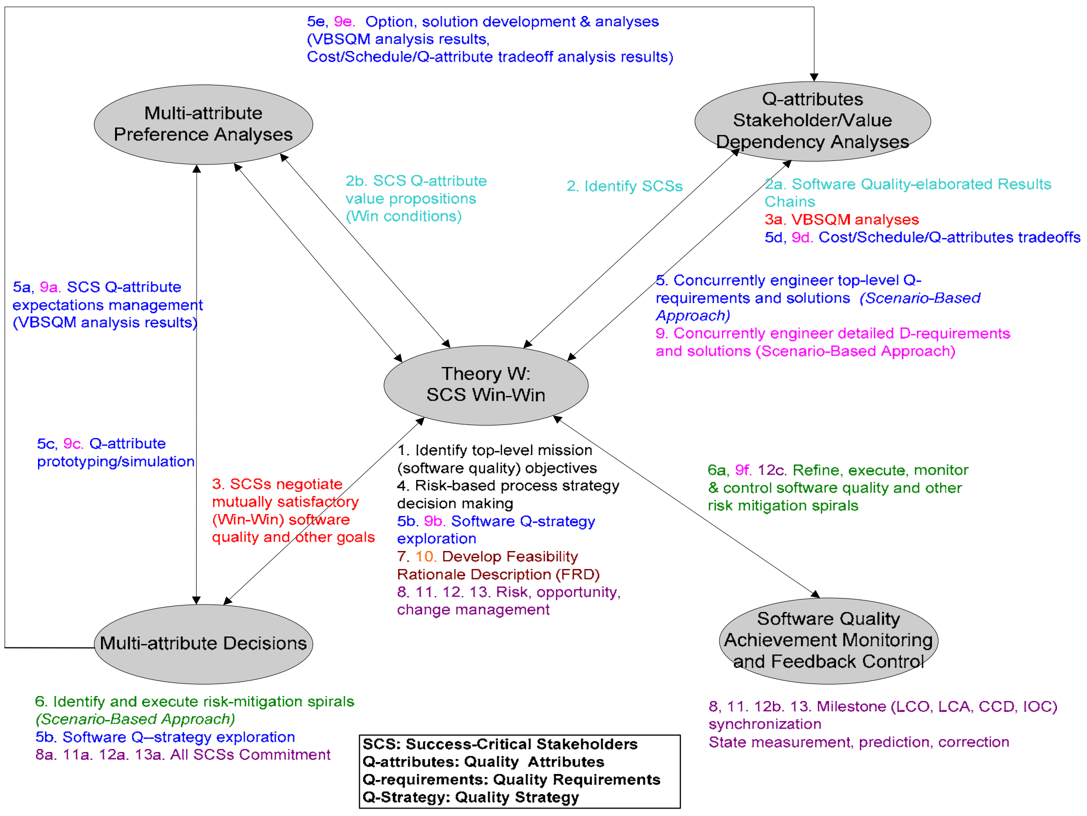

5e, 9e. Option, solution development & analyses
(VBSQM analysis results,
Cost/Schedule/Q-attribute tradeoff analysis results)
Multi-attribute Preference Analyses
Q-attributes Stakeholder/Value Dependency Analyses
2b. SCS Q-attribute value propositions (Win conditions)
2. Identify SCSs
2a. Software Quality-elaborated Results Chains
3a. VBSQM analyses
5d, 9d. Cost/Schedule/Q-attributes tradeoffs
5a, 9a. SCS Q-attribute expectations management (VBSQM analysis results)
5. Concurrently engineer top-level Q-requirements and solutions (Scenario-Based Approach)
9. Concurrently engineer detailed D-requirements and solutions (Scenario-Based Approach)
Theory W: SCS Win-Win
5c, 9c. Q-attribute prototyping/simulation
3. SCSs negotiate mutually satisfactory (Win-Win) software quality and other goals
1. Identify top-level mission (software quality) objectives
4. Risk-based process strategy decision making
5b. 9b. Software Q-strategy exploration
7. 10. Develop Feasibility Rationale Description (FRD)
8. 11. 12. 13. Risk, opportunity, change management
6a, 9f. 12c. Refine, execute, monitor & control software quality and other risk mitigation spirals
Multi-attribute Decisions
Software Quality Achievement Monitoring and Feedback Control
6. Identify and execute risk-mitigation spirals (Scenario-Based Approach)
5b. Software Q–strategy exploration
8a. 11a. 12a. 13a. All SCSs Commitment
8, 11. 12b. 13. Milestone (LCO, LCA, CCD, IOC) synchronization
State measurement, prediction, correction
SCS: Success-Critical Stakeholders
Q-attributes: Quality Attributes
Q-requirements: Quality Requirements
Q-Strategy: Quality Strategy

Figure 3.17. Mapping of VBSQA Process into "4+1" VBSE Theory

Figure 3.17 presents the mapping of top-level VBSQA process steps to the "4+1" Theory of VBSE. The ovals represent the quality-oriented theories. Each process step serves as either input or output of a particular theory.

Chapter 4

VBSQM Application and Results

This chapter presents the application of the VBSQM in different value situations. Section 4.1 compares the initial VBSQM ROI analyses on two different types of software projects. Section 4.2 shows the VBSQM ROI analysis on different operational scenarios/software classes in NASA/USC Inspector SCRover project. Section 4.3 presents the VBSQM combined risk analyses in three representative project business cases. Section 4.4 compares the ROI of value-based testing techniques with value-neutral testing techniques and shows the overall risk reduction of value-based testing based on VBSQM combined risk analyses. The application results validate the following two research hypotheses:

- **Hypothesis 1**: VBSQM can be used to determine how much software quality investment is enough in different value situations.
- **Hypothesis 2:** Assuming non-linear value functions (e.g., Pareto distribution) are used, value-based software quality achievement techniques improve project return on quality investments and reduce the overall project risks.

4.1 VBSQM ROI Analyses for Different Types of Software Systems

4.1.1 A Dependable Order Processing System

This section illustrates an initial use of the VBSQM to develop a rough software quality Return On Investment (ROI) analysis, using the Sierra Mountainbikes order processing system business case analysis in [Boehm-Huang 2003]. It is a case study representative of two real world order processing systems. We uses their business case analysis as the baseline for assessing future investments in software quality over and above the nominal investments usually made for business data processing systems.

4.1.1.1 Business Case Analysis of Sierra Mountainbikes Order Processing System

Table 4.1 summarizes the business case for an improved order processing system through its proposed development in 2004-2005 and proposed operation in 2005-2008. The cumulative cost of fully replacing the old system by the new one is $6 million, of which $3.44 million is for software. Table 4.1 also shows the corresponding expected dollar benefits and return on investment, ROI = (Benefits – Costs) / Costs, annually for the years 2004-2008. For simplicity in this analysis, the costs and benefits are shown in 2004 dollars to avoid the complications of discounted cash flow calculations, and the 10% annual growth rate in estimated market size is not compounded, both for simplicity and conservatism.

As seen in columns 2-5 of Table 4.1, Sierra's current market share and profit margins are estimated to stay roughly constant over the 2004-2008 period, with annual profits growing from $7M to $12M, if the new program is not executed. This is a conservative estimate, as the problems with the current system would increase with added sales volume, leading to decreased market share and profitability.

The next columns in Table 4.1 up through ROI show the expected improvements in market share and profit margins (due both to economies of scale and decreased operational costs) achievable with the new system, and the resulting ROI relative to continuing with the current system. They show that the expected increase in market share (from 20% to 30% by 2008) and profit margins have produced a 45% ROI by the end of the second year of new-system operation (2006):

$$ROI = \frac{Benefits - Costs}{Costs} = \frac{9.4 - 6.5}{6.5} = 0.45$$

The expected ROI by the end of 2008 is 297%.

The final four columns in Table 4.1 show that qualitative as well as quantitative aspects of ROI need to be estimated and tracked. They show a simple balanced-scorecard array of expected 2004-2008 improvements in overall customer satisfaction and three of its critical components: percentage of late deliveries, ease of use, and in-transit visibility. The latter capability was identified as both important to distributors (if they know what is happening with a delayed shipment, they can improvise workarounds), and one which some of Sierra's competitors were providing. Sierra's expected 2004-2008 improvements with the new system were to improve their 0-5 satisfaction rate on in-transit visibility from a low 1.0 to a high 4.6, and to increase their overall customer satisfaction rate for order processing from 1.7 to 4.6.

	Current System			New System											
				Financial								Customers			
Date	Market Size ($M)	Market Share %	Sales	Profits	Market Share %	Sales	Profits	Cost Savings	Change in Profits	Cumulative Change in Profits	Cumulative Cost	ROI	Late Delivery %	Customer Satisfaction (0-5)	In-Transit Visibility (0-5)	Ease of Use (0-5)
12/31/03	360	20	72	7	20	72	7	0	0	0	0	0	12.4	1.7	1.0	1.8
12/31/04	400	20	80	8	20	80	8	0	0	0	4	-1	11.4	3.0	2.5	3.0
12/31/05	440	20	88	9	22	97	10	2.2	3.2	3.2	6	-.47	7.0	4.0	3.5	4.0
12/31/06	480	20	96	10	25	120	13	3.2	6.2	9.4	6.5	.45	4.0	4.3	4.0	4.3
12/31/07	520	20	104	11	28	146	16	4.0	9.0	18.4	7	1.63	3.0	4.5	4.3	4.5
12/31/08	560	20	112	12	30	168	19	4.4	11.4	29.8	7.5	2.97	2.5	4.6	4.6	4.6

Table 4.1. Order Processing System: Expected Benefits and Business Case

4.1.1.2 VBSQM Software Quality ROI Analysis and Results

The VBSQM ROI analysis begins by analyzing the effect of increasing software quality investments from the normal business levels to the next higher levels of investment in analysis tool support, peer review practices and test thoroughness. The resulting increase from a Nominal to High COCOMO II RELY rating level increases the mean time between failure (MTBF) from 300 hours to 10,000 hours (see Figure 3.2). And it also incurs a $3.44M * (1.10-1.0) = $344K additional software quality investment.

Assuming from relevant business experience a mean time to repair of 3 hours yields an improvement in availability = MTBF/(MTBF + MTTR) from 300/303 ≅ 0.99 to 10,000/10,003 ≅ 0.9997. If we use availability as a proxy for software quality, and assume that a 1% increase in downtime is roughly equivalent to a 1% loss in sales, we can use the Sierra Mountainbikes business case to determine a software quality Value Estimating Relationship (VER). Applying the difference between a .01 loss in sales and a .0003 loss in sales to the 2005-2008 Sierra new system sales total of \$531M (adding up the 2005-2008 numbers in column 7 of Table 4.1) yields a net return on the software quality investment of (.01) (\$531M) – (.0003) (\$531M)= \$5.31M – 0.16M = \$5.15M. The COCOMO II Cost Estimating Relationships (CER's) for Tool Support and Process Maturity also generate software rework savings from the investments in early defect prevention and removal of \$0.45M, for a total savings of \$5.59M. The resulting software quality ROI is (5.59 – 0.345) / 0.345 = 15.1.

A related interesting result is that added quality investments have relatively little payoff, as there is only \$0.16M left to be saved by decreasing downtime. Table 4.2 summarizes the VBSQM ROI analysis results by increasing the availability level of the Sierra Mountainbikes Order Processing System from Nominal to High, High to Very High and Very High to Extra High, respectively.

4.1.1.3 Project Key Stakeholders' Feedback and Implications

In discussing this analysis in non-directed interviews with relevant business personnel (including a representative of one of the failed projects on which the Sierra

Mountainbikes case study was based), they indicated that the results were realistic, and that use of the tool would have helped them in several decision situations.

However, when asked whether their interests in software quality would disappear with the negative ROI shown in Table 4.2 in going from a High to a Very High RELY level, some of the respondents indicated that availability would no longer be a concern, but that other concerns such as security would likely be deserving of further investment once the availability was not causing significant business losses. This would suggest that they are operating within a Maslow need hierarchy in which satisfied availability needs are no longer motivators, but in which higher level needs such as reducing security risks may now become more significant motivators.

This casts the analysis of software quality (Q-) attributes in an entirely new light. Previously, the problem of software attribute analysis has been largely cast as an exercise in static multi-attribute optimizing or satisficing, operating on some pre-weighted combinations of Q-attribute satisfaction levels. The practical decision-making issue above indicates that achieving an acceptable or preferred combination of software Q-attributes leads to a new situation in which the attribute priorities are likely to change.

In this situation, software Q-attribute requirements become more emergent than pre-specifiable. The process for achieving acceptable software Q-attribute requirements becomes no longer a single-pass process, but an evolutionary process, subject to the need to anticipate and develop architectural support for downstream software quality needs.

4.1.2 A Mission Critical NASA Planetary Rover

As part of the NASA High Dependability Computing Program (HDCP), we also performed an exploratory VBSQM analysis of a representative NASA planetary rover robot; in collaboration with Jet Propulsion Laboratory (JPL) Mission Data System and Mars Science Laboratory personnel.

4.1.2.1 Business Case Analysis of NASA Planetary Rover

JPL planetary mission experts indicated that a planetary rover's top-priority software Q-attribute is survivability. Otherwise, its first failure on a remote planet will be its last. Survivability has some attributes such as graceful degradation, so that at a minimum, the rover can keep enough power and communication capability to be able to transmit its status to its Mission Control Center on Earth, and to receive and execute recovery commands from the Mission Control Center. However, since availability is strongly correlated with survivability, and is more straightforward to analyze, we used availability as a proxy for survivability.

A more detailed hazard analysis and fault-tolerance/graceful degradation cost-benefit analysis would be needed for safety or survivability ROI analyses. A more detailed vulnerability and protection cost-benefit analysis would be needed for security ROI analysis.

As part of a business case for determining a planetary rover software quality value estimating relationship (VER), we used a total mission value equal to a representative planetary rover mission cost of $300 million, with a baseline software cost of $20M at a Nominal COCOMO II software reliability (RELY) rating. We also assumed as a baseline that a 1% decrease in availability was roughly equivalent to a

1% loss in the mission value of $300M. For a representative mean time to repair, the JPL personnel indicated that 150 hours or roughly a week was a representative amount of time for a Mission Control Center to diagnose a planetary rover problem, to formulate and prepare a recovery sequence, and to test its validity.

4.1.2.2 VBSQM Software Quality ROI Analysis and Results

Since a planetary rover requires at least a High level of reliability and availability, the VBSQM ROI analysis begins by analyzing the effect of increasing reliability and availability investment from High to Very High which corresponds to the COCOMO II estimates of a cost increase of 16% to $25.2M and an increase in mean time between failure (MTBF) from 10,000 hours to 300,000 hours (see Figure 3.2). Assuming a mean time to repair (MTTR) of 150 hours yields an improvement in availability = MTBF/(MTBF+MTTR) from 0.985 to 0.9995 as shown in Table 4.2.

Further increasing RELY rating from Very High to Extra High goes outside the COCOMO II rating scale range and requires a special analysis. Investing in an independent verification and validation (V&V) activity to bring the MTBF up to 1 million hours (114 years), using the COQUALMO Extra High levels of activity, incurs estimated additional investments in formal analysis tool support and usage ($2M), peer review practices ($1M), and test thoroughness ($3M). As a cross-check, the resulting $6M investment is near the usual value of 30% added cost for Independent V&V on such missions. At this point, the added reliability and availability investments have negative payoff (ROI = -0.98), as there is only $0.15M left to be saved by decreasing downtime. Table 4.2 summarizes the VBSQM ROI analysis results by increasing the availability level of the Planetary Rover from High to Very High and from Very High to Extra High, respectively.

4.1.2.3 Project Key Stakeholders' Feedback and Implications

In discussing these results in a non-directed group discussion with the JPL personnel, they indicated that such tool and analysis results made them more confident in deciding an appropriate software availability investment level for their project situation.

They also pointed out that the losses in reputation, corrective action, and in some cases human lives as with the Columbia Shuttle failure, can amount to much more lost value than the cost of the mission, and that these can produce a positive ROI for an Extra High availability investment effort.

4.1.3 Comparing the VBSQM Availability ROI Analysis Results of Order Processing System and NASA Planetary Rover

Figure 4.1 summarizes and compares the VBSQM availability ROI analysis results of the Sierra Mountainbikes Order Processing System and the NASA Planetary Rover. The trend of the Order Processing System is in a black dashed line and that of the Planetary Rover is in a red solid line.

Thus we see that different mission situations lead to different diminishing returns points for the business application, whose ROI goes negative in going from a High to a Very High RELY rating; and for the planetary rover application, whose positive ROI is sustained through Very High, but not through Extra High .

Note also that we have assumed a linear value model in this analysis, which results in the monotone decreasing of the mission value by decreasing the availability level. Most often, value models look like S-shaped economic production functions in Figure 3.5, with an initial low-slope Investment segment, a high-slope High Returns segment, and a final low-slope Diminishing Returns segment.

Projects	RELY Rating	MTBF (hrs)	MTTR (hrs)	Availability	Loss ($M)	Increased Value ($M)	Cost ($M)	Δ Cost ($M)	ROI
Sierra Mountain-bikes Order Processing System	Nominal	300	3	0.9901	5.31	0	3.45	0	----
	High	10K	3	0.9997	0.16	5.15	3.79	0.344	14
	Very High	300K	3	0.99999	0.005	0.155	4.34	0.55	-0.72
	Extra High	1M	3	1	0	0.005	5.38	1.04	-1.0
NASA Planetary Rover	High	10K	150	0.9852	4.44	0	22	0	----
	Very High	300K	150	0.9995	0.15	4.29	25.2	3.2	0.32
	Extra High	1M	150	0.99985	0.045	0.105	31.2	6.0	-0.98

Table 4.2. VBSQM Availability ROI Analysis Results of Sierra Mountainbikes Order Processing System and NASA Planetary Rover: Increasing MTBF

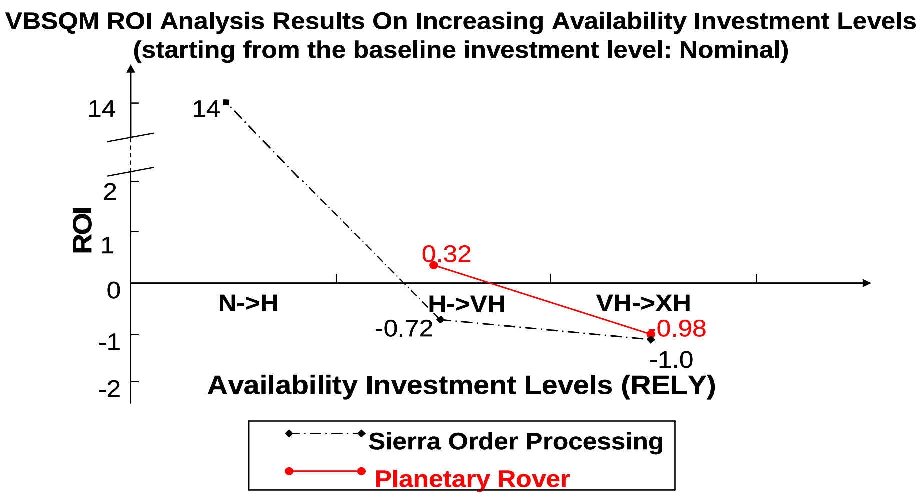


Figure 4.1. Comparing the VBSQM Availability ROI Analysis Results of Sierra Mountainbikes Order Processing System and NASA Planetary Rover

4.2 VBSQM ROI Analyses for Different Scenarios in One Software System

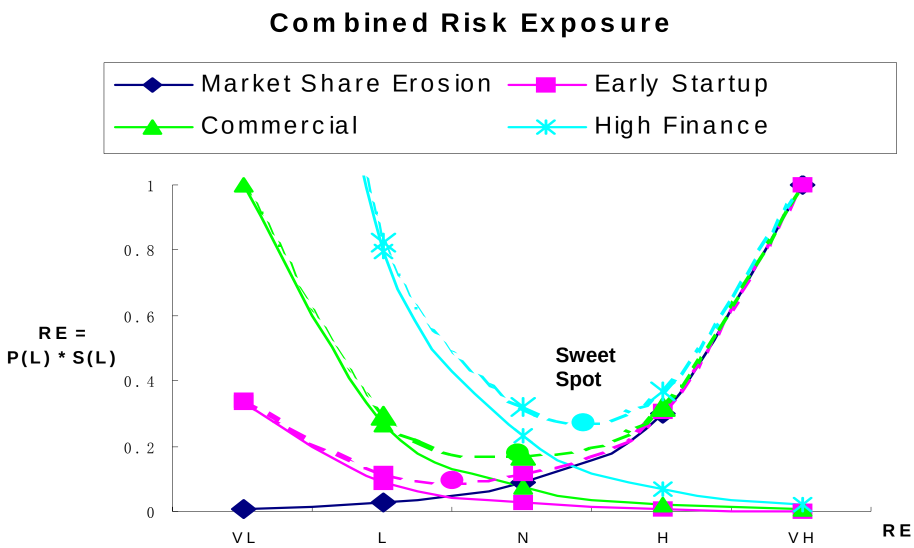
We performed VBSQM ROI analyses for three different classes of mission scenarios in NASA/USC Inspector SCRover project. The analysis results helped stakeholders to define three different Availability levels for the SCRover project. Please refer to section 5.1.2.3.1 for details.

4.3 VBSQM Combined Risk Analyses to Determine How Much Software Quality Investment is Enough

Figure 4.2 demonstrates how to build up information and perform VBSQM combined risk analyses (refer to section 3.2.2.2) based on the COCOMO II, COQUALMO, and VERs illustrated in section 3.2.1.4. The analyses results show what level of quality investment is enough for three typical project business cases (i.e. early start-up, normal commercial, and high finance) [Huang-Boehm 2006].

The probability of loss $P_q(L)$ (e.g., financial, reputation, future prospects) due to unacceptably low quality can be estimated based on the COQUALMO estimate of delivered defect density in Figure 3.4: to first order, the fewer the defects, the lower the probability of loss. We can use the Very Low estimate of 60 defects/KSLOC in Figure 3.4 as the baseline for $P_q(L)$, and set its default value to 1. The $P_q(L)$ for other RELY ratings from Low to Very High can then be computed based on the corresponding delivered defect density relative to the baseline, as shown in the second row of numbers at the bottom of Figure 4.2. A baseline VER for the size of loss *$S_q(L)$* due to unacceptable quality can be obtained for value-based testing [Bullock 2000, Gerrard 2002, Ramler et. al. 2006] from the Pareto distribution in Figure 4.3(a), using a negative Pareto distribution for value loss as shown in rows 3, 4 and 5 at the bottom of Figure 4.2. In Figure 4.2, relative *$S_q(L)$* is shown in three representative business cases such as early start-up (row 3), representing relatively defect-tolerant early adopters; normal commercial (row 4), representing the Bullock data [Bullock 2000]; and high finance (row 5), representing very high-volume time-sensitive cash flows dependent on reliable operation of the software system. For simplicity, we use a factor of 3 to distinguish the relative values of the three cases. Then we can compute the software quality investment risk exposure as $RE_q = P_q(L) \times S_q(L)$.

These values enable us to calculate relative software quality investment risk exposures as functions of added testing time for the three classes of business cases. Each of these classes of stakeholders can then determine their own “how much software quality investment is enough?” sweet spot by combining their software quality investment risk exposure curve with their market share erosion risk exposure curve RE_m (obtained from the Critical Region of the market share loss curve in Figure 3.10) shown as the line of diamonds in Figure 4.2. For simplicity, we have shown this to be equal to 1.0 for a Very High RELY rating and an added COCOMO-calibrated 54% delay in time to market, and decreasing by a factor of 0.3 for each successively lower RELY rating, as shown in the bottom line of numbers in Figure 4.2.



	VL	L	N	H	VH	RELY
COCOMO II:	**0**	**12**	**22**	**34**	**54**	**Added % test time**
COQUALMO:	**1.0**	**.475**	**.24**	**.125**	**.06**	**$P_q(L)$**
Early Startup:	**.33**	**.19**	**.11**	**.06**	**.03**	**$S_q(L)$**
Commercial:	**1.0**	**.56**	**.32**	**.18**	**.10**	**$S_q(L)$**
High Finance:	**3.0**	**1.68**	**.96**	**.54**	**.30**	**$S_q(L)$**
Market Risk:	**.008**	**.027**	**.09**	**.30**	**1.0**	**RE_m**

Figure 4.2. Combined Risk Exposures: Early Startup, Commercial and High Finance

Finally, we can find a sweet spot (the minimum) from the combined risk exposure due to both unacceptable software quality and market share erosion. Figure 4.2 shows the three combined *RE* curves in dashed lines and the corresponding oval sweet spots of software quality investment levels for the three business cases. For the high finance business case, its sweet spot of software quality investment is located at the right-most side because the risk exposure of low system quality RE_q dominates. For the early startup business case, its sweet spot of software quality investment located at the left-most side because the risk exposure of high market share erosion RE_m dominates. Such risk analyses can help project decision-makers determine where is the optimal stopping point in planning for "how much testing will be enough," or more generally, the optimal level of the software quality investment for their project based on their own business case.

The baselining at 1.0 of the highest mainstream size of loss due to low software quality and of the highest risk exposure due to market share erosion means that the model in Figure 4.2 can be straightforwardly adapted to other business situations. For example, a software vendor in the High Finance market sector could replace the 1.0 baseline market share risk exposure with his/her estimate of a $10M loss in late delivery of a new feature in row 6 of Figure 4.2 by multiplying the numbers in row 6 by $10M. Similarly, he/she could adjust the numbers in row 4 by replacing the 1.0 baseline business loss size in row 4 by his/her estimate of a $30M business loss of releasing a Very Low quality upgrade, to generate a curve similar to the star curve in Figure 4.2 with a RELY investment sweet spot halfway between Nominal and High. Note that other analyses can be made to determine how much software quality investment is enough for other types of mission value loss reference points or alternative curves [Boehm-Huang 2004a]. We should point out that determining absolute business values such as $10M and $30M may not be easy, particularly if one has not done a business case for the project. However, even relative values can be used to obtain useful decision insights.

In addition, with VBSQM combined risk analyses, users can perform sensitivity analyses of the most appropriate quality investment level and strategies with respect to uncertainties in stakeholder value propositions or marketplace conditions, for different risk exposure situations, or for additional qualitative considerations.

The current VBSQM tool supports the software quality ROI analysis discussed in section 4.1 and 4.2 as well as the above combined risk analyses. As for the combined risk analysis, it provides the default values of $S_q(L)$ and RE_m of each RELY rating for three business cases (i.e., early start-up, normal commercial and high finance). Users can also provide their own values for $S_q(L)$ and RE_m based on their project business case. After the user inputs the project size in SLOC and rates each COCOMO II cost driver except RELY according to their own project situation, VBSQM will generate the curve for combined risk exposure and help to locate the sweet spot for their software availability investment level. In addition, since the VBSQM tool is spreadsheet-based, it is easy to modify to handle other types of analyses using different VERs for different situations, or to perform analyses of the sensitivity if the outcomes or sweet spots to unavoidable uncertainties in the input parameters or value functions.

4.4 Value-based Testing vs. Value-neutral Testing

Much of current software testing research and tool building is done in a value-neutral setting, in which every requirement, object, test case, and defect is considered equally important. This section compares the value-based testing and value-neutral testing in terms of the ROI and combined project risks.

4.4.1 Value Estimating Relationships (VERs) and ROI Analysis

With value-neutral testing, such as using the output of most automated test generation (ATG) tools, the earned mission value with invested testing effort will be linear shown as the dotted line in Figure 4.3(a), since each requirement and test case is considered equally important. However, in most operational situations [Bullock 2000, Gerrard 2002, Ramler et. al. 2006], the value earned by each requirement will more likely follow a Pareto distribution shown as a solid curve in Figure 4.3(a). Value-based testing focuses the testing effort on the roughly 20% of the features that provide roughly 80% of the system value. As an example from Bullock's project experience [Bullock 2000], the Return On Investment (ROI) analysis is based on the following assumptions:

- $1M of the development costs have been invested in the customer billing system by the beginning of testing.
- The ATG tool will cost $300K and will reduce test costs by 50% as promised.
- The business case for the system will produce $4M in business value in return for the $2M investment cost.
- The business case will provide a similar 80:20 distribution for the remaining 14 customer types.

Table 4.3 shows the relative levels of investment costs, business benefits, and returns on investment ROI = (benefits – costs) / costs, for the value-neutral ATG testing and value-based Pareto testing strategies. Figure 4.3(b) provides a graphic comparison of the resulting ROIs. As seen in Figure 4.3(b), the resulting ROI for value-based testing peaks earlier and at a considerably higher level than that for value-neutral testing.

Bullock's empirical data shows that testing each customer type improves billing revenues from 75% to 90%, and that a single one of the 15 customer types account for 50% of all billing revenues. The ROI analysis results indicate that focusing initial testing on that one customer type provides an immediate boost in billing revenues per dollar invested in testing.

% of Test Runs	**Value-neutral ATG Testing**				**Value-based Pareto Testing**			
	Cost	**Value**	**Net Value**	**ROI**	**Cost**	**Value**	**Net Value**	**ROI**
0	1300	0	-1300	-0.10	1000	0	-1000	-1.0
10	1350	400	-950	-0.70	1100	2560	1460	+1.33
20	1400	800	-600	-0.43	1200	3200	2000	1.67
40	1500	1600	100	+0.07	1400	3840	2440	1.74

Table 4.3. Comparative Business Cases: ATG and Pareto Testing

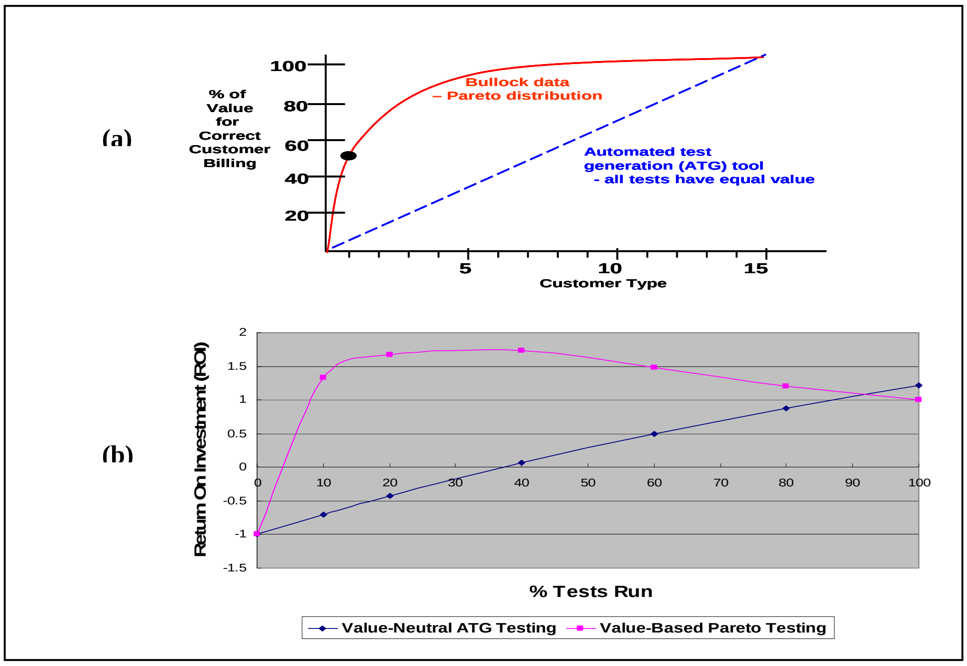
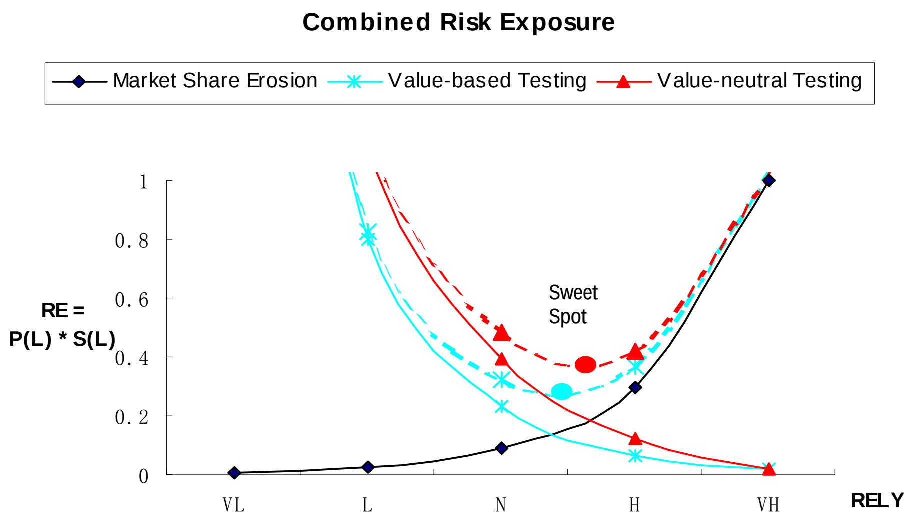


Figure 4.3. (a) Value Estimating Relationships (VERs) for Value-Neutral Testing vs. Value-Based Testing; (b) Return On Investment (ROI): Value-Neutral ATG Testing vs. Value-Based Pareto Testing

However, there may be good reasons, such as preserving good customer relationships, to continue testing after reaching the peak ROI. And frequently the testing experience from the high-value testing can be used to make the selection of test cases or the use of test data generators for the lower-value requirements more cost-effective.

4.4.2 VBSQM Combined Risk Analyses

VBSQM combined risk analyses can also be applied in comparing the value-based quality achievement techniques with value-neutral ones in terms of the combined project risks. Figure 4.4 presents the VBSQM combined risk analysis results of value-based testing and value-neutral testing, when the high finance business case is used as an example. The decrease in $S_q(L)$ with testing time will be linear for the value-neutral testing, while the decrease in $S_q(L)$ with testing time will follow the negative Pareto distribution for the value-based one as shown in rows 3 and 4 at the bottom of Figure 4.4. The combined risk exposure of value-based testing is shown as the dashed line of triangles, while the combined risk exposure of value-neutral testing is shown as the dashed line of stars in Figure 4.4. The sweet spot of value-neutral testing moves to up and right of that of value-based testing, which is also shown in Figure 4.4. For this example, the minimum risk exposure for value-neutral testing is about 40% higher than that of value-based testing. Of course, the project will need to invest in some form of early requirements prioritization, such as business case analysis, stakeholder win-win negotiation, Total Quality Management, or agile methods story prioritization, but these generate other project advantages as well.



COCOMO II:	0	12	22	34	54	Added % test time
COQUALMO:	1.0	.475	.24	.125	.06	$P_q(L)$
Value-Based:	3.0	1.68	.96	.54	.30	$S_q(L)$: Pareto
Value-Neutral:	3.0	2.33	1.65	0.975	.30	$S_q(L)$: Linear
Market Risk:	.008	.027	.09	.30	1.0	RE_m

Figure 4.4. High Finance Combined Risk Exposures: Comparing Value-Based Testing vs. Value-Neutral Testing

4.5 Conclusions

This chapter describes how the integrative VBSQM was applied to perform ROI analyses and combined risk analyses in order to determine an approximate quantitatively-optimal software quality investment level and strategy for a project. (There may be qualitative factors such as key-stakeholder satisfaction that may make other levels or strategies preferable.) With such value-based software quality models, users can perform sensitivity analyses of the most appropriate quality investment level and strategies with respect to uncertainties in stakeholder value propositions or marketplace conditions, for different risk exposure situations, or for additional qualitative considerations. The combined risk analysis model realized in VBSQM is also valuable for determining the relative payoff of value-based vs. value-neutral testing, which can be up to 40% higher for high-value applications, as shown in Figure 4.4.

Even with only approximate information on relative values, the models can provide a framework to help reason about quality investment tradeoffs and decisions.

Chapter 5

VBSQA Process Application and Results

This chapter presents the application of Value-Based Software Quality Achievement (VBSQA) process framework in three case studies. Section 5.1 discusses the entire procedure of applying the VBSDA process in NASA/USC Inspector SCRover (ISCR) project developed by six graduate students in the Center for Software Engineering (CSE) at USC. The final product served as one of the testbeds for NASA High Dependability Computing Program (HDCP). Section 5.2 describes the partial application of VBSDA process in a Graduate Software Engineering class e-service project for USC librarian. Section 5.3 shows how we apply VBSDA process framework in a real world Enterprise Resource Planning (ERP) project in China. These application results validate the following research hypothesis:

- **Hypothesis 3**: The Value-Based Software Dependability Achievement (VBSDA) Process can be applied by rational decision-makers to
  - **3a.** determine whether a software system with stakeholder mutually satisfactory software quality (Q-) attribute requirements is achievable.
  - **3b.** help stakeholders and projects avoid software Q-attribute mismatches and achieve successful software quality outcomes.

5.1 Apply VBSDA Process in NASA/USC Inspector SCRover (ISCR) Project

This section describes how the VBSQA process was applied in NASA/USC Inspector SCRover (ISCR) project to help success-critical stakeholders achieve successful software quality outcome.

5.1.1 NASA/USC ISCR Project Overview

The ISCR project was developed to serve as a distributable HDCP testbed for evaluating current and emerging dependability-enhancing technologies. It involved obtaining requirements from the USC Department of Public Safety (DPS) for an autonomous robot that could investigate the possible presence of hazardous materials in an environment unfit or dangerous for human intervention. Such an environment could be caused due to an earthquake or a failed chemical/biological experiment in a chemistry/biological laboratory. It would have several risks, such as loss of human health or life due to failure to identify a dangerous target with chemical leak or radiation, and the damage of robot itself.

Here are the top-level requirements that were determined for the ISCR system. The robot shall be able to autonomously maneuver around in the area designated by the robot operator and identify the potentially hazardous targets. The robot shall simultaneously return pictures taken by the camera mounted on the robot and the available sensor information to the designated host computer. Additionally, the robot shall maintain enough power so that it can return back to the initial designated location.

The development of the SCRover was planned in 3 increments. The case study is based on the increment 3 of the project which covers most of the important mission scenarios.

5.1.2 ISCR Application of VBSDA Process

5.1.2.1 Identify Top-level Mission Objectives and Stages

The top-level objectives and stages for the Inspector SCRover (ISCR) are summarized in section 5.1.1. And the top-level capabilities of the SCRover are as follows:

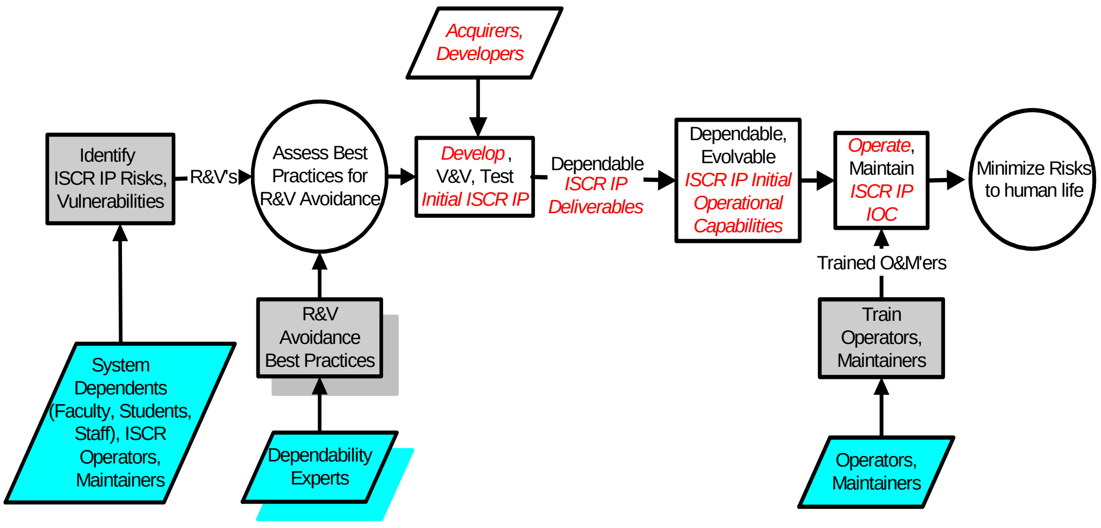
- Autonomously maneuver around in the area designated by operator and identify potentially hazardous targets
- Return continuous camera video images and sensor information to host computer
- Power maintenance

We applied the VBSDA process on the increment 3 of the ISCR project.

5.1.2.2 Perform Project Cost/Benefit Analysis

Besides project budget estimation, project success-critical stakeholders and their top-level value propositions are identified to perform the project benefit analysis. The Results Chain technique, developed by the DMR Consulting Group [Thorp 1998] is a way to identify missing initiatives and success-critical stakeholders in a software development project.

Figure 5.1 shows the software quality-elaborated Results Chain for developing the Initial Operational Capabilities (IOC) of the ISCR increment 3 operational scenarios. We have omitted the Assumptions for simplicity, but added the identification of success-critical stakeholders in parallelograms. Note that the text in italic shows the original simple initial Results Chain for the project developing the Initial Operational Capability (IOC) of the ISCR increment 3 information processing (IP) and operational capabilities without the software quality considerations. The full Results Chain identifies additional success-critical Initiatives, such as prevention and avoidance of ISCR risks and vulnerabilities (R&Vs), training operator and maintainers. Besides the Acquirers and Developers identified in the simple initial Results Chain, the additional software quality initiatives identify success-critical stakeholder class (Dependability Experts), and also the employment of additional software quality enhancing tools and techniques such as verification and validation (V&V). Other success-critical stakeholders are also identified whose inputs are needed for the risk and vulnerability analysis: ISCR System Dependents (i.e. USC Lab Faculty, Students and Staff), Operators and Maintainers.



Acquirers, Developers
Identify ISCR IP Risks, Vulnerabilities
R&V's
Assess Best Practices for R&V Avoidance
Develop, V&V, Test Initial ISCR IP
Dependable ISCR IP Deliverables
Dependable, Evolvable ISCR IP Initial Operational Capabilities
Operate, Maintain ISCR IP IOC
Minimize Risks to human life
Trained O&M'ers
R&V Avoidance Best Practices
Train Operators, Maintainers
System Dependents (Faculty, Students, Staff), ISCR Operators, Maintainers
Dependability Experts
Operators, Maintainers

Figure 5.1. Software Quality-Elaborated Results Chain for ISCR Increment 3

5.1.2.3 Stakeholders Negotiate Quality (and Other) Goals and Relevant Mission Scenarios

Table 5.1 identifies a matrix of the primary ISCR success-critical stakeholders as rows and their prioritized goals with respect to the ISCR system development, operation, and evolution as columns. The specific columns represent the primary categories of system requirements to be negotiated by the stakeholders.

Project goals and requirements include desired constraints on the ISCR project such as choices of programming language, infrastructure packages, and computing platforms; development and operational standards; and constraints on budgets, schedules, and other scarce resources as listed in Table 5.2. *Capability* goals include the functions the SCRover should perform. *Interface* goals include message formatting and content, and interaction protocols with other interoperating systems. *Level of Service* goals include the ISCR quality goals, except for cost and schedule (covered under Project goals) and interoperability (covered under Interface goals).

Evolution goals include downstream goals that the initial ISCR architecture should support, such as deferred capabilities or scalability to accommodate workload growth.

Goals / Stakeholders	Project	Capability	Interface	Level of Service	Evolution
ISCR System Dependents (USC lab faculty, students, staff, etc.)			Tab	le	
ISCR Operators					
ISCR Acquirers (USC DPS)	Table 5.2				
ISCR Developers, Maintainers (USC CSE)					

Table 5.1. Inspector SCRover (ISCR) Stakeholder/Goal Matrix (Priorities: High, Medium, Low)

Goals / Stakeholders	Project Goals	Priority
ISCR System Dependents (USC lab faculty, students, staff, etc.), Operators	Develop an autonomous mobile robot that shall help the USC DPS perform its goals of investigating hazardous agents in the USC labs.	H
	Post-Mission data analysis	M
ISCR Acquirers (USC DPS)	Acquire the Core Initial Operational Capabilities (IOC) within budget and schedule	H
ISCR Developers, Maintainers (USC CSE)	Develop IOC within $200K and 9 months	H
	Use MDS (Mission Data System) Framework	H

Table 5.2. Inspector SCRover (ISCR) Stakeholder/Goal Matrix I: ISCR Project Goals and Priorities (Priorities: High, Medium, Low)

Stakeholders / Goals	ISCR System Dependents/Operators
Project	…
Capability	…
Interface	…
Level of Service	H: Availability >= 0.9998 for ISCR mission critical scenarios H: Availability >= 0.993 for ISCR on-line operational scenarios M: Availability >= 0.807 for ISCR post-mission data analysis scenarios H: Accuracy of Target Sensing >= 99% …
Evolution	…

Table 5.3. Inspector SCRover (ISCR) Stakeholder/Goal Matrix II: ISCR System Dependents/Operators Goals and Priorities (High, Medium, Low)

Instead of using one number to define the ISCR system availability goal, we distinguished three classes of mission scenarios. As shown in Table 5.3, the ISCR system dependents' and operators' goals for system availability are 0.9998 for mission-critical scenarios, 0.993 for on-line operational scenarios, and 0.807 for post-mission data analysis scenarios. Such numbers are traditionally difficult to determine. We will show how the Value-Based Software Quality Model (VBSQM) [Boehm-Huang 2004b] helps determine them in the following subsection.

5.1.2.3.1 Determine ISCR Availability Goals: VBSQM ROI Analysis

Multiple stakeholder negotiation of ISCR system goals involves a mix of collaborative win-win option exploration with prototyping and analysis of candidate options. Here, the VBSQM can be used to help the stakeholders determine how much availability is enough for the three primary classes of ISCR scenarios. Table 5.4 shows the key availability-related parameters for the software related to the three classes of ISCR scenarios; the size in thousands of source lines of code (KSLOC), the cost per line of code and total cost independent of investments in software reliability, and the dollar mission value of risk if the class of the scenarios fails. For instance, there are 15 KSLOC of software for mission-critical scenarios such as *Target Sensing* and *Target Rendezvous*. Its cost per instruction of a Nominal COCOMO II Required Reliability level is \$6.24/LOC (at graduate-student labor rates), leading to a nominal cost of \$93.6K. A failure in the mission-critical software is likely to cause complete contamination and replacement of the robot and the lab, with an impact equal to the \$2.5M of an entire lab. A failure and loss of availability of the online-operational ISCR scenarios (i.e., display continuous video images and sensor data to operator) would require repair and rerun of the mission, possibly losing $200K of lab equipment. A failure of post-mission data analysis would require debugging, fixing, and regression testing the software, typically costing about $14K.

Table 5.5 summarizes an VBSQM analysis of the return on investment (ROI) involved in increasing the reliability level from Nominal to High; High to Very High; and Very High to Extra High. As determined from the calibrated parameters in the COCOMO II [Boehm, et. 2000], and COQUALMO [Steece, et. 2002] models on which VBSQM [Boehm-Huang 2004a] is based, increasing the reliability level of the ISCR On-Line Operational software from Nominal to High involves an additional $45K(0.10) = $4.5K investment. It results in an increase in MTBF from 300 to 10,000 hours, which at an experienced-based Mean Time To Repair (MTTR) of 72 hours results in an increase in availability from .807 to .993. Using a linear relation between fraction of downtime and fraction of lost value as in [Demillo 2001], this 0.186 increase applied to the $200K risk impact of the On-Line Operational scenario results in an added benefit of $200K (0.186) = $37.2K, and a resulting ROI = (37.2-4)/4 = 7.29. However, an additional $45K (0.16) = $7.19K investment to take the software from High to Very High only gains in a ($200K)(.9998-.993)=$1.38K benefit, for a negative ROI of -0.81.

Classes of Scenarios	Size (KSLOC)	Nominal		Risk Impact ($K)
		$/LOC	$K	
Mission-Critical	15	6.24	93.6	2500
Online-Operational	8	5.62	45	200
Post-Mission Data Analysis	6	4.48	26.9	14

Table 5.4. Size, Cost, and Risk Impact of Three Classes of SCRover Scenarios

COCOMO RELY Level	Nom	High	Very High	Extra High
MTBF(hrs)	300	10,000	300,000	1,000,000
Availability (MTTR=72hrs)	.807	.993	.9998	.99993
Incremental Availability		.186	.0069	.00001
Incremental Cost Fraction		0.10	0.15	0.24
Mission-Critical				
Incr. Cost @ $$93.6K		$.9.36K	$14.55K	$16.84K
Incr. Benefit @ $2.5M		$466K	$17.27K	$.42K
ROI = (B-C)/C		+48.8	+0.15	-0.98
Online-Operational				
Incr. Cost @ $45K		$4.5K	$7.19K	
Incr. Benefit @ $200K		$37.28K	$1.38M	
ROI = (B-C)/C		+7.29	-0.81	
Post-Mission Data Analysis				
Incr. Cost @ $26.9K		$2.69K	$4.3K	
Incr. Benefit @ $14K		$2.61K	$100	
ROI = (B-C)/C		-0.03	-.0.98	

Table 5.5. VBSQM ROI Analysis of ISCR Increment 3 Availability Goals for Three Scenario Classes

This and the ROI results for the other two classes of ISCR scenarios calculated in Table 5.5 are summarized in Figure 5.2. The incremental cost of achieving the higher availability levels still keeps the total cost below $200K. From a pure calculated ROI standpoint, one could achieve some potential savings by interpolating to find the availability-requirement levels at which the ROI goes from positive to negative, but it is best to be conservative in a safety-related situation. Or one can identify desired and acceptable availability levels to create a tradeoff space for balancing availability with other software quality attributes.

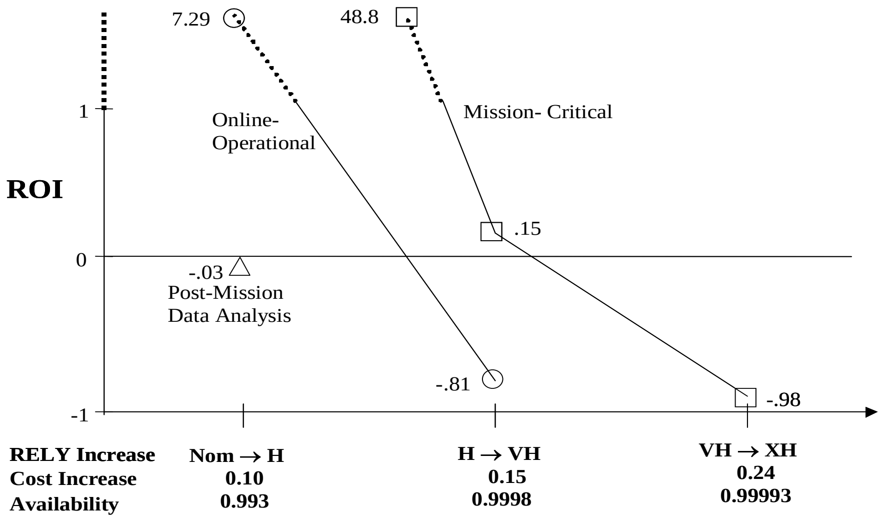


Figure 5.2. Summary of VBSQM ROI Analysis of ISCR Increment 3 Availability Goals

5.1.2.4 Concurrently Engineer Top-level Q-attribute and Other Requirements and Solution Tradeoff Spaces; Identify Top-level Risks and Execute Risk-mitigation Spirals

E1. Identify software quality (Q-) attributes

The top-level software Q-attributes for the ISCR system are availability, safety, accuracy, performance, usability, cost and schedule.

E2. Establish system operational profile scenarios and prioritize scenarios

The operational scenarios of the ISCR increment 3 Initial Operational Capability (IOC) identified by the stakeholders are shown in the Table 5.6. The lower-priority scenarios can be added in past-IOC increments.

Scenarios	Component Scenarios	Priority
Target sensing		H
Target rendezvous	Trajectory planning	H
	Localization	
	Obstacle avoidance	
Display environment state information to operator	Return target state info state variable to operator	H
	Return terrain state variable to operator	
Display sensor and actuator health state information to operator	Return camera state variable to operator	H
	Return range finder health state variable to operator	
	Return wheel motor health state variable to operator	
	Return battery state of charge to operator	
Display continuous camera video images to operator		M
Post-mission data analysis		L
Goal conflict identification and resolution		L

Table 5.6. ISCR Increment 3 Operational Profile Scenarios

E3. Map Q-attributes into scenarios and determine metrics, stakeholder/value dependencies and value estimating relationships (VERs) for software Q-attributes of each scenario

Table 5.7 shows the top-level direct stakeholder/value dependencies on the Q-attributes in the *Target Sensing* scenario. Acquirers and developers are not directly concerned with availability, accuracy, safety, etc., but become concerned with them when their operational stakeholders are.

Stakeholders / Q-attributes	System Dependents	Operators	Acquirers	Developers	Maintainers
Availability	*	**			*
Accuracy		**			
Cost			**	*	
Schedule			**	**	
Evovability					**

** Critical	* Significant	( ) Insignificant or indirect

Table 5.7. *Target Sensing* Scenario: Stakeholder/Value Dependencies on software Q-attributes

E4. For each scenario, stakeholders define their acceptable and desired values for concerned Q-attributes

The results of the VBSQM ROI analysis for three ISCR scenario classes discussed in section 5.1.2.3.1 were used as a guidance for stakeholders to define their expected and desired levels for Q-attributes based on the priority of a particular scenario and their value dependencies on the scenario-related Q-attributes.

In both *Target Sensing* and *Target Rendezvous* scenarios, Operators have critical dependency on availability so that they defined the 99.98% acceptable availability for those two scenarios, which is corresponding to the Very High RELY cost driver rating in the VBSQM ROI analysis. System Dependents and Operators are most concerned about the lab safety which is directly dependent on the accuracy of *Target Sensing* so that they proposed a requirement of 99% acceptable and 99.9% desired accuracy of target sensing.

E5. For each scenario, identify the risks of not achieving the acceptable values of Q-attributes

E6. For each scenario, identify the architecture/technologies to mitigate the Q-risks

E7. Architecture/technologies evaluation

By grouping the risks for the two mission-critical ISCR scenarios *Target Sensing* and *Target Rendezvous*, the top-level risks for not achieving the 99.98% acceptable availability requirements are summarized in Table 5.8. Similarly, the risks for not achieving the 99% acceptable accuracy requirement in the *Target Sensing* scenario are summarized in Table 5.9.

For each pair of risk and architecture/technology strategy in each scenario, we provide an estimate (expert judgement) of the mitigation of the risk in the scenario as shown in the “Mitigation” column in Table 5.8 and Table 5.9. We define the “mitigation” as the proportion by which the risk would be reduced were that architecture/technology strategy to be applied. At the same time, the cost/effort of applying a particular architecture/technology strategy should also be recorded.

E8. Identify conflicting Q-attributes and perform tradeoff analysis

In the ISCR project, the conflicting Q-attributes we experienced are primarily due to the confliction between the cost/effort of achieving the acceptable levels of some Q-attributes and the project cost/schedule constraint. Note that in our stakeholder/value dependency framework, cost and schedule are also considered as the Q-attributes which both system acquirers and developers directly depend on.

Based on the stakeholder acceptable levels of availability and accuracy requirements in E4 and the architecture/technology evaluation results in E7, developers and ISCR acquirers found that cost and schedule constraint would be overrun if all the technology strategies need to be applied for the risk mitigation. Specifically, in the *Target Rendezvous* scenario, in order to satisfy the 99.98% acceptable availability requirement, we would need to replace the current Laser Range Finder (LRF) with one having a wider range and develop more complex algorithms to accommodate the two environment risks (i.e., mirrors and big holes). On the other hand, in order to satisfy the 99% acceptable accuracy of *Target Sensing*, a redundant camera would need to be installed on the SCRover to mitigate the risk of inaccurate camera data.

The development team consisting of 5 half-time CSE research assistants was required to finish the increment 3 of the ISCR project within 9 months. For the *Target Rendezvous* scenario, it would incur almost 25% of the schedule overrun in order to design, implement and test the more complex algorithms to accommodate the two environment risks. In addition to that, a more expensive LSR would need to be installed which will incurs budget overrun. As for the achieving of 99% acceptable accuracy requirement in the *Target Sensing* scenario, it only incurs the budget overrun if a redundant camera needs to be installed. Then three options listed as below were proposed for stakeholders to perform the tradeoff analysis:

- **Option 1.** Achieve 99.98% availability in *Target Rendezvous* but sacrifice other D-attributes such as accuracy, performance and usability, etc.
- **Option 2.** Relax both 99.98% availability and 99% accuracy requirements to some extent.
- **Option 3.** Relax the schedule and cost constraint and try to achieve both 99.98% availability and 99% accuracy requirements.

Based on the assessment of the above options on a group of prioritized Q-attributes with scenarios, stakeholders could assess their tradeoff functions and score the three options. Finally the option 2 won.

Categories	Risk Description	Impact on Scenario(s) S(L)	Prob. of Loss P(L)	Architecture/Technology Strategies	
				Name	Risk Mitigation
Requirement	Missing availability requirement	0.5	0.1	Peer Review	0.99
	Ambiguous or untestable requirement	0.5	0.8	Peer Review	0.95
Design/Code	Communication problems between components (e.g., parameter type mismatch, pre/post condition mismatch, etc.)	0.8	0.3	MDS Framework	0.3
				Peer Review	0.5
				Mae	0.9
				Acme Studio	0.7
				Unit Testing	0.5
				Integrated Simulation	0.8
	Incorrect method/function/algorithm	0.9	0.5	Peer Review	0.8
				Mae	0.6
				Acme Studio	0.6
				Unit Testing	0.8
				Integrated Simulation	0.9
	Data initialization errors	0.3	0.2	Peer Review	0.6
				Unit Testing	0.8
				Integrated Simulation	0.9
Environment	Reflective obstacles (e.g. mirrors)	0.5	0.01	New LRF	0.5
				Reflective obstacle avoidance algorithm	0.95
	An environment with big holes	0.9	0.05	New LRF with a wider range	0.9
				Hole avoidance algorithm	0.9

Table 5.8. ***Target Sensing*** **and** ***Target Rendezvous*** **Scenarios: Top-level Risks of Not Achieving 99.98% Availability and Risk Mitigation of Architecture/technology Strategies**

Categories	Risk Description	Impact on Scenario(s) S(L)	Prob. of Loss P(L)	Architecture/technology Strategies	
				Name	Mitigation
Hardware	Inaccurate camera data	0.8	0.3	Sensor Redundancy	0.99
Design/Code	Inaccurate estimation of target position/safety status	0.8	0.3	MDS Framework	0.8
				Peer Review	0.5
				Mae	0.6
				Acme Studio	0.3
				Monte Carlo Simulation	0.9

Table 5.9. ***Target Sensing*** **Scenario: Top-level Risks of Not Achieving 99% Accuracy and Risk Mitigation of Architecture/technology Strategies**

E9. Stakeholders negotiate WinWin balanced Q-attribute requirements and redefine the acceptable and desired values for conflicting Q-attributes

The Operator agreed to relax the availability requirement by ignoring the mirrors and big holes because the probability of their occurrence in the DPS mission modes is very low as shown in Table 5.8. It was redefined as the acceptable availability of *Target Rendezvous* in a controlled environment (without mirrors or big holes) is 99.98%. As for the accuracy requirement, developers proposed a new approach to ensure at least 98% accuracy of *Target Sensing* by double-checking the target sign from two different angles. It not only avoids the installation of a redundant camera but also saves some software development effort. Both System Dependents and Operators accepted this suggestion and relaxed the acceptable accuracy of *Target Sensing* to 98%.

5.1.2.5 Develop Initial Feasibility Rationale; Hold Life Cycle Objective (LCO) Review

A Life Cycle Objective (LCO) Review was held with the participation of all the project key stakeholders, and independent experts who were NASA Jet Propulsion Laboratory (JPL) planetary mission software experts for ISCR. This indicates a milestone of the LCO phase in the WinWin Spiral Model. The exit criteria of LCO Review were to provide at least one feasible architecture to satisfy the requirements, and to provide proofs of requirement satisfaction including the software quality requirements. At the same time, the initial risk analyses identified all the major risks and propose an initial risk mitigation plan. Risks without mitigation in LCO stage had to be resolved in Life Cycle Architecture (LCA) stage.

The result of LCO ARB was to *Pass* and to go to Step 9 (see Table 3.4). However, a risk was identified that the tool evaluation needs for the HDCP tool researchers had been incompletely defined for ISCR.

5.1.2.6 Concurrently Engineer Detailed Q-attribute and Other Requirements and Solutions; Resolve Risks

Thus, the major new activity in Step 9 (see Table 3.4 in section 3.3.2) involved surveying HDCP interventionists for additional evaluation needs. The primary need identified was a three-dimensional graphic user interface (3D GUI).

Originally, developers planned to use Player/Stage as the robot simulator platform. Because of the 3D GUI requirement, we had to reevaluate the existing technologies or identify new technologies to mitigate this Q-risk. In this case, since Stage doesn’t support a 3D GUI, the developers had to find a replacement. After the evaluation, stakeholders finally chose Gazebo because it supports the new stereo camera model and a 3D GUI which also enabled most devices to be directly controlled/inspected through the simulator GUI. Since Stage and Gazebo are both Player-compatible, client programs written using one simulator can usually be run on the other with little or no modification. The key difference between these two simulators is that whereas Stage is designed to simulate a very large robot population with low fidelity, Gazebo is designed to simulated a small population with high fidelity [Gazebo 2005]. Thus Gazebo fits with most of DPS missions which can be accomplished by a few robots. Furthermore, Gazebo is more valuable to stakeholders since it improved the usability and evolvability of the system.

5.1.2.7 Develop Detailed Feasibility Rationale; Hold Life Cycle Architecture (LCA) Review

The Life Cycle Architecture (LCA) Review was held with the participation of all the project key stakeholders and the JPL experts. The exit criteria of Life Cycle Architecture Review Board (LCA ARB) were to commit one architecture to satisfy all the requirements of the system. And the LCA FRD provided the detailed proofs of all requirement satisfaction including the software quality requirements.

The result of the LCA ARB was again to *Pass*, and to proceed to Step 12 (see Table 3.4).

5.1.2.8 Construct, Test, and Deploy System

During the construction phase of ISCR Increment 3, the proposed mission scenarios discussed in section 5.1.2.4 were used to simulate the ISCR Q-attribute requirements (e.g. availability, accuracy) in Gazebo and to evaluate whether the acceptable Q-attribute levels can be achieved. ISCR Core Capability Demo (CCD) was successfully performed with the participation of JPL experts and other project key stakeholders.

5.2 Avoid Q-attribute Mismatches in Fulltext Title Database Project

This section illustrates how the VBSQA process helped avoid software Q-mismatches through a USC e-service project, the Fulltext Title Database (FTD).

Note that only the process activities closely related to avoidance of Q-attribute mismatches are discussed.

5.2.1 Fulltext Title Database (FTD) Project Overview

The Fulltext Title Database (FTD) system is designed and developed for faculty, students, researchers and librarians at USC. USC subscribes to many different vendors' databases that index and abstract popular and scholarly articles and make them available over the World Wide Web. In some cases these databases include the full text of the articles as well. A major problem faced by faculty, students, researchers and librarians is identifying where a particular periodical title is indexed, the dates of coverage and whether it is available in full text. Title lists are available from the vendors, but are large files that can be difficult to use on the fly. As a result, the initiative of this project was to retrieve and consolidate the information in vendors' title lists into a searchable database that can be accessed via the web. A major challenge is to architect the software system to be completed by satisfying the project key stakeholders' requirements (including software quality requirements) with 24 weeks. The following sections describe how the process elements (as indicated by the section titles) were applied within the project lifecycle to avoid Q-attribute mismatches.

5.2.2 Stakeholder Identification and Feature Prioritization

The key stakeholders of the FTD project included a system acquirer representative from USC Information Service Division (ISD), a USC-ISD librarian who is designated as database administrator and system maintainer, and software developers from USC-CSE. The USC/GroupSystem EasyWinWin requirements negotiation tool were used to help stakeholders converge on a mutually satisfactory (win-win) set of project and capability requirements (named as system features here). The resulting prioritization is presented in Table 5.10. Stakeholders also identified the top-priority quality requirements associated with the system features, which include architecture Evolvability and Schedule.

System Features		Priority
F1	Search and locate full-text journal titles by title keywords	Very High
F2	Search and locate full-text journal titles by title keywords and date, title keywords and volume or number, title keywords with any combination of the other three attributes	Very High
F3	Provide hyperlinks to vendors' databases in the searching results	Very High
F4	Update Fulltext Title Database using current vendors' title lists	High
F5	Automatically FTP downloaded title lists from administrator's local machine to remote server	High
F6	System administrator authentication	High
F7	Administrator password maintenance	High
F8	Add new vendor's title list profile	Medium
F9	Delete existing vendor's title list profile	Medium
F10	Modify existing vendor's title list profile	Medium
F11	View existing vendor's title list profile	Medium
F12	Allow more searching options starting with searching by ISSN, etc.	Low

Table 5.10. Prioritized Fulltext Title Database System Features

5.2.3 Engineer Software Quality Requirements and Tradeoff Analysis

5.2.3.1 Estimate Features for Firm 24 week Schedule

The student developers participate in the project course for a maximum of two semesters, hence there is a firm 24-week schedule constraint. Exacerbating this, students are not full time developers (i.e. not 40 hours/week developers) and this must be taken into account. The system construction phase mostly occurs in the second semester, which leaves only 12 weeks for implementation and transition. The project allocated 2 weeks at the end of the semester to insure successful transition to the customer, leaving only 10 weeks for actual implementation. It is assumed that student will spend no more than 12 hours per week doing project work. Thus there are about (10 wks) (12 project hrs/week) = 120 project-hrs. available per team member. To use COCOMO II for effort estimates we had to convert student development effort to COCOMO II person-months [Boehm et. al. 2000a]. For this purpose this we assumed that 72% of the effort is spent in construction (28% elaboration of system concept) and only 66% of the day is spent doing project work. As such, 120 project-hrs is approximately (1.67)(.72)(.66 project-hrs/total hrs)(152 hrs/person-month) and so with our five team members our COCOMO II effort should not exceed (5) (1.67)=8.35 person-months or around 34 person-weeks.

The developers used COCOMO II to determine that prioritized features from the EasyWinWin negotiations indicated in Table 5.10 would take at least 40 person-weeks using the pessimistic 90% confidence limit on effort. This significantly exceeds the 34 person-week schedule constraint, hence we must consider dropping some features. COCOMO II estimated that the seven top-priority features F1-F7 could be implemented pessimistically in 17 person-weeks. However there is no guarantee that implementing these features will result in a usable system. This will be elaborated in the next section.

5.2.3.2 Core Capability Determination

Next, a core capability feature set from Table 5.10 was determined. To accomplish this, developers first had to make sure that the core capability set was selected so that its features add up to a coherent and workable end-to-end operational capability. Second, the remainder of the lower-priority IOC requirements and subsequent evolution requirements must be used in determining a system architecture that will facilitate evolution to a full operational capability. For the Fulltext Title Database project, three core capabilities, comprised of multiple top-priority features were proposed:

1. <u>Provide a full-text journal title search capability</u>

   This core capability aggregates the first three highest priority features F1, F2, and F3. At the same time, a low priority feature F12 was assigned to this capability and thus it could be dropped (although still architected for) if needed.

2. <u>Update the Fulltext Title Database</u>

   This core capability incorporates three high priority features F4, F5 and F6.

3. <u>Administrator password maintenance</u>

   This uses the high priority feature F7 which enables the system administrator to change his/her password in the future and was deemed critical for long term functionally.

The above three capabilities formed a coherent, end-to-end core capability (as negotiated with the stakeholders) that, if implemented would also provided reasonable value to users. While it is not totally coincidental that these happen to be the top-7 priority features, this was by no means assured until after this core capability analysis and negotiation with stakeholders. As discussed in section 5.2.3.1, COCOMO II estimated that even pessimistically these features could be developed in 17 person-weeks. As it turned out, the developers did indeed have enough time to add most of the lower prioritized capabilities and even a late requirements change for the IOC, in spite of the originally pessimistic 40 person-week estimate. The key lesson here is that this effort was strategically risk managed so that there was high assurance of delivering a functionally valuable system for the IOC. This was clearly facilitated by designing an underlying architecture that was easy to scale up to the full feature set and workload requirements after the core capabilities were developed. This will be elaborated next.

5.2.3.3 Determine Architecture Evolvability: Evolvability and Schedule Tradeoff Analysis

Architecting a set of core capabilities to encapsulate foreseeable sources of changes within modules incurs overhead. It is not enough to simply modularize the core capabilities; the sources of anticipated changes must also be taken into account. This includes accounting for the possible new features themselves (e.g. method stubs, abstract objects), their interaction with each other (e.g. interfaces), and their interaction with the existing core capabilities (for example changes to method parameters and return types, pre- and post-validation). The architecture must be strategically designed in such a way as to enable delivery of the core features while providing an appropriate level of Evolvability without introducing an excessive risk of schedule overrun. Too much Evolvability may delay the delivery of the core capabilities. Too little may not enable changes to be made after core capability delivery without substantial re-work. In general, haphazard Evolvability architecting can lead to excessive schedule risk.

We illustrate the points above by making use of a "combinatorial" effort model shown as Formula 5.1. While this model can be empirically calibrated for a particular project, it does not strictly provide a predictive estimation of effort. Rather, it indicates the "expectation" (or average) of additional effort that may be incurred for making changes within the architecture of a given degree of Evolvability. As such, the model can help with the advanced planning of architecture with respect to specific schedule constraints. Suppose we plan to build a software system with N features, and we plan for architectural Evolvability by grouping the features into modules. As a first order estimation to this problem we assume that each module on the average will implement m features (so m is at least 1 and cannot exceed N). We make no assumptions on the actual distribution of the features. For small systems or in localized parts of a system, the distribution may have a significant effect on the modification effort. However taken as a whole over a sufficiently complex system, the average is reasonably representative. In larger systems, some modules may have many features, thereby reducing the amount in others, or perhaps they are equally distributed. In either case, the overall effort in making changes to a group of these modules is "expected" (that is on the average) to be the same. We call m the modularity factor and so we expect to create N/m modules (some modules may be empty, but are defined so as to accommodate features later). The extreme case $m=N$

would be a "totally rigid" architecture, or all features in a single module, whereas at the other extreme $m=1$ is "hyper-flexible", where every feature exists in a single module. Consider the expected change effort E to add or modify a fraction α of the N total planned features after the core features have been developed. We must account for the effort to add an expected αN features, create some new modules, create new interfaces and interfacing with old modules, and interfacing between new and old features. The following model is a first order "expectations" model to represent this effort:

$$E = c_1 \alpha N + c_2 \binom{N/m}{2} + c_3 \binom{\alpha m}{2} N/m + c_4 \alpha (1-\alpha) m N \qquad (5.1)$$

E – total change effort
N – total number of features
m – modularity factor
α – a fraction of anticipated changes
c_1 – average effort to develop a feature
c_2 – average development effort of per inter-module interaction
c_3 – average development effort of per intra-module feature interaction
c_4 – average effort to accommodate a new feature within each existing feature in a module

A little elaboration of the above model is in order here. The first term is the effort for adding or modifying αN features. The value c_1 represents the average effort to develop a feature. One possible means of establishing this value is by taking the core capability COCOMO II schedule estimate and dividing by the number of core capabilities. In addition, we should also consider adjusting the value by adjusting some of the relevant COCOMO II cost drivers to account for the experience gained and established development. The other constants in the equation may be estimated empirically in other ways. Note that the core capability COCOMO

II schedule estimate does not incorporate the effort for planning the architecture design to add or modify features up to *N*.

The second term accounts for the additional effort required to develop the modules for all *N* planned features (e.g. interfaces and all possible interactions between them) and c_2 represents the average inter-module interaction development effort per module.

The third term represents the effort to make changes within each module to accommodate the new (or modified) features (e.g. passed values, validations, etc.) with each other. Note that this is where modularity helps. The fewer features there are per module, the lower this term is. Again, the c_3 coefficient represents the average effort per intra-module feature interaction.

The final term is slightly more subtle as it accounts for the effort needed to accommodate the new features within the existing features for each particular module. Within each module an average of αm features will be modified and these changes may affect an average of $(1-\alpha)m$ of the unmodified features. The product of these provides the average number of interactions. The coefficient c_4 represents the average effort to accommodate each new feature within each existing feature in a module. Note that it is reasonable that this term contributes its maximum at $\alpha = 1/2$ as more or fewer changes would imply less interaction effort between modified and existing features. Typically this accounts for the type of changes due to interfaces and data validation. Here again, the fewer features per module, the lower this term becomes.

The individual terms described above hopefully account for all the major sources of change effort. But why is it reasonable to assume the terms are additive? Certainly there are strong dependences between the effort sources. However, here is where we take advantage of the “linearity of expectations” whereby the sum of the averages is the average of the sum regardless of dependencies. Optimizing over an average value is a common useful modeling approach and for our purposes practical in light of the linearity simplification.

The values of the effort coefficients c_2, c_3 and c_4 relative to c_1 will vary as a function of the relative modularity and coupling involved in the application, as approximately represented by such techniques as design structure matrices [Baldwin-Clark 2000, Sullivan et. al. 2001]. An example of these coefficients from our case study described in section 5.2.1 where N=12 is $c_1 = 1.7, c_2 = 0.4, c_3 = 0.7, c_4 = 0.6$ (An explanation on how we estimated these values is detailed below in this section.). We use these values in Figure 5.3 to illustrate the change effort E versus fraction modified α for various modularity values m:

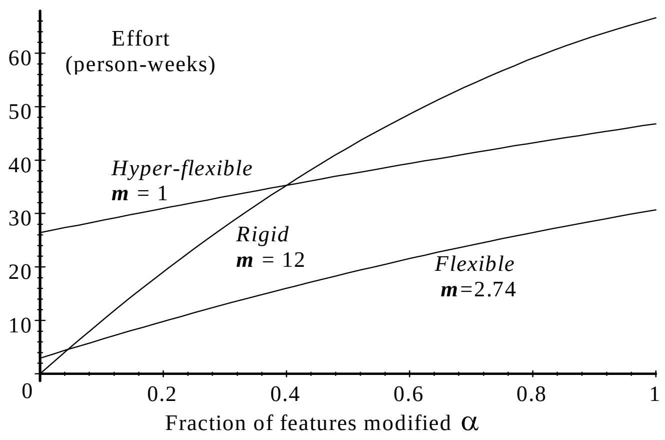


Figure 5.3. Change Effort E vs. Fraction of Features Modified α

The general trend shown in Figure 5.3 indicates that the initial effort for a hyper-flexible architecture is very high, while the rigid architecture is very low, but increases significantly when features are added. For our case study objective of adding 5 features between the Core Capability Demo and IOC milestones ($\alpha = 5/12$), we find that a modularity factor of 2.74 minimizes the change effort in our model. This degree of Evolvability not only allows us to adapt to changes well but also has low initial design effort. So long as the initial effort does not exceed our schedule constraint, it is feasible to strategically plan for future changes to features using this modularity. It is clear from this example that careless modularization can lead to excessive, perhaps fatal additional project effort. A good example of a hyper-flexible architecture was the MEDLARS II publication system. It had seven layers of abstraction and was finally dropped after two years and four hardware upgrades could not handle the workload. A good example of a totally-rigid architecture was the initial design of the ARPANET Interface Message Accessor software, which was so tightly coupled that it could not be modified even for performance measurement.

It is important to note that our model deals with determining a degree of Evolvability that helps achieve a particular schedule constraint (which will be exemplified further in the case study section) and not determining the best modular design according to the natural dependencies of the capabilities and techniques. There is related work that does deal with this more tactical issue by considering the net-present-value of modularization within a design structure matrix [Sullivan et. al. 2001]. After using such an approach to determine a modular architecture, our approach can help determine if it or some variation is feasible with respect to the risk of exceeding a particular schedule constraint. Alternatively, it can be used in advance of design modularization to help narrow the design option space by indicating a schedule-feasible range of the number of modules and average functions per module. Beyond this, the model does not provide any technical design guidance predictive estimates.

Here we estimate how flexible the architecture should be to accommodate the 12 features for the Fulltext Title Database system based on the expected effort model. We previously determined that there are 3 core capabilities integrating features F1-F7. As such we anticipated a desire to add 5 medium or low priority features after the core capabilities were developed, hence for our model, α = 5/12. The average effort per feature c_1 (=1.7) was estimated based on another COCOMO II estimate of 11.9 person-weeks for the 7 core feature set by adjusting the COCOMO II cost drivers for having the experience of already building the core-capabilities. The values $c_2(=0.4), c_3(=0.7), c_4(=0.6)$ were determined empirically from actual effort expended on the prototype and expert judgement. Naturally we would also like to architect the system for ease of adding the remaining lower priority features to get as many of the 12 features as possible. This was also important as the clients' requirements were volatile. For example, after an early project review, they proposed a new requirement that would enable the Fulltext Title Database system to accommodate vendors' title lists in various formats. It aggregated the four medium-priority features F8, F9, F10, F11. Clients also promoted F12 from a low-priority evolutionary requirement up to high priority.

Given the above information, if the developers had planned to build a hyper-flexible architecture, for example by creating generic database "adaptors" to enable the system to be compatible with an assortment of databases, the schedule estimate from the model gives nearly 53.6 person-weeks. Note that even though the effort needed to add features is very low, the effort expended on both planning to add/modifying features in the hyper-flexible architecture and actually adding any of the remaining 5 features is estimated at around 36.6 person-weeks (see Figure 5.3). While undoubtedly our model is inaccurate, it does indicate that there is a considerable risk of exceeding the 34 person-week schedule limit. On the other hand, if we had used a rigid architecture, the 36.5 person-weeks estimate there is also undue schedule risk to re-architect and implement the new features along with the core capabilities.

Therefore, the developers tried to build a flexible, but not "too flexible" architecture given possible requirement changes in the following manner. Given that we anticipated 5 feature changes, we found that m = 2.74 in our model gives the minimum effort (using the usual calculus method of setting the derivative to zero) for the value α = 5/12. This resulted in an effort estimate of about 33.5 person-weeks (17 person-week for the core feature set based on a COCOMO II estimate, plus 16.5 person weeks "change effort" which is the additional effort E for adding 5 features, see Figure 5.4). Based on this estimate, it's possible for the developers to meet the 34 person-week schedule constraint. Subject to particular technical considerations for modularity design, if we target architecting 3 or 4 modules which grouped 4 or 3 coherent features in each (on the average), we increase our chances of being able to deliver the 5 additional features beyond the core within the 34 person-week schedule. This degree of modularity is nice in that it has a reasonably low initial effort (only slightly higher than a rigid architecture) and that adding features beyond the 5 doesn't increase the effort rapidly. Such an architecture tolerates volatile requirements well, which proved to be the case for this project.

With this in mind, we designed four modules: the Fulltext Title Search module, the FTD System Administrator module, and the Fulltext Title Information Archive module and the Fulltext Maintenance module which was initially a "stub" used to encapsulate the four medium-priority features. Thus the actual modularity factor *m* was 3, which was consistent with our estimated "optimal" modularity factor 2.74 from the model. While mainly the technical considerations drove the modularization (and hence the choice of four modules) that this was compatible with the strategic estimate provided two valuable project management elements. First, it helped rationalize our invested effort (and the plan to do such) for the design of the four modules. Second, it increased our confidence in our decision to be satisfied with only four modules, helping suppress the urge to "generalize" the design further. Both these factors contributed to managing the risk of not meeting our fixed delivery schedule.

Several general interfaces (such as Title_Search_Information) were created to accommodate future changes and capabilities. In addition, some of these modules utilized method stubs and meta-methods to adhere to these interfaces without implementing the capabilities (but enable easy implementation at a later time). At the completion of the project, we ended up implementing the 5 remaining features in addition to several requirement changes.

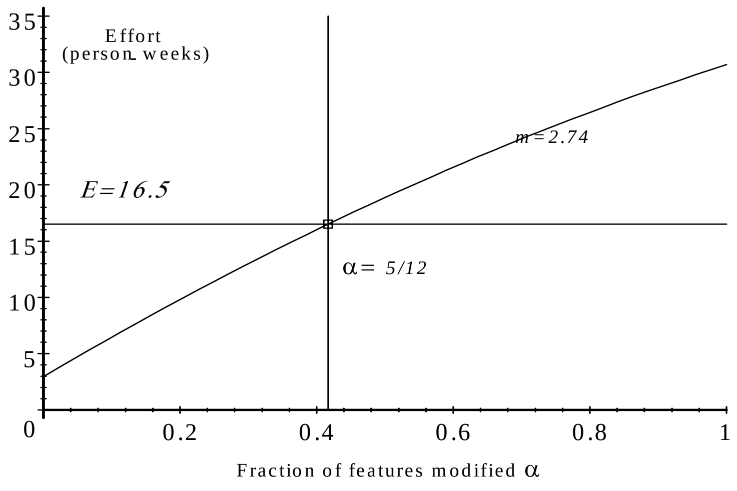


Figure 5.4. Change Effort E When 5/12 Planned Features Modified

5.2.4 Project Results and Discussion

Based on the tradeoff analysis between FTD architectural Evolvability and project schedule, we strategically planned two iterative increments to develop the FTD system development and to cope with the potential requirement changes within 24 weeks. Three FTD core capabilities were successfully delivered within approximate 70-80% of the project schedule in increment I. In addition, we also successfully accommodated two new high-priority change requests from clients in increment II as follows:

1. Accommodate vendors’ title lists in various formats
2. Add F13 (Partial keyword searching) and drop F12 (Allow more searching options starting with searching by ISSN, etc.)

Software develops have long understood the negative effects of too little architecture Evolvability. Many have also understood that attempting to achieve maximum Evolvability can pose the risk of not leaving sufficient time for project completion, of increasing project cost beyond acceptable limits and/or rendering the desired software quality impossible to achieve given cost and schedule constraints. Previous attempts at achieving a balance between too little and too much architecture Evolvability have been based solely on technical considerations. Our current approach to strategic architectural Evolvability has been used successfully on 24 of 26 e-services applications at USC, to deliver highly client-satisfactory applications on a fixed schedule in a climate of rapid changes. We have been using and refining this approach to assist project managers in determining an appropriate degree of architectural Evolvability with respect to fixed schedule, cost, or quality constraints by introducing a modularity factor for the software architecture based on the core capabilities and a set of anticipated changes. In particular, architectural Evolvability determination is an integral part of the Schedule as Independent Variable (SAIV) approach [Boehm-Brown 2001, Boehm et. al. 2002].

While our approach does not predict technical actual design modularity, we have found it instrumental in guiding technical modularity design decisions and managing risk of schedule overruns. It shows an example of how to construct a tradeoff function between software Evolvability and project schedule. An analogous approach can be used to provide guidance to meet cost, quality, or cost-schedule-quality goals.

5.2.5 Extension to Cost, Quality, and Schedule/Cost/Quality Goals

Simply substituting "cost" for "schedule" in the steps described above provides an equally effective way to handle cost rather than schedule constraints. Handling combined Schedule/Cost/Quality constraints is a straightforward extension of this. It involves setting the system's quality goals (e.g., a delivered defect density of 0.3 nontrivial defects per thousand source lines of code (KSLOC), or of 0.03 nontrivial defects per function point), and tracking progress with respect to achieving the desired combination of schedule, cost, or quality goals. If any of these goals becomes unachievable in delivering the current feature set, the project must drop enough lower-priority features to make the combination of goals achievable. There may be limits to the project's ability to do this, such as insufficient schedule to develop even a viable core capability, which we have discussed via a production-function perspective in [Boehm-Brown 2001].

5.3 Apply VBSDA Process in Real-world ERP Software Development

5.3.1 Introduction

Enterprise Resource Planning (ERP) is a business management system that integrates all facets of the business process, including planning, manufacturing, sales, and marketing. The booming economy in China has encouraged the development of ERP software to improve various business operations such as inventory control, order tracking, customer service, finance and human resources. Attracted by the potential huge profit, more and more software companies are jumping into this field, which leads to severe competition among ERP solution providers. They are expected to provide continuous value realization for their success-critical stakeholders to survive the competition. These trends drive the changes in the characteristics of ERP software development in China as follows:

- A variety of stakeholders with different value propositions are involved in the entire software development life cycle.
- Product lines are maintained as a basis for future upgrades.
- Three process patterns (deadline-driven, product-driven and market-trend driven) are selectively applied in their software development life cycle based on different business cases.
- Different software quality assessment criteria are set based on different business cases by various project success-critical stakeholders. Thus different software development activities are adopted in different process patterns.

ERP solution providers in China have established their own sets of software development process activities. Most of the activities are based upon value-neutral approaches and/or the Waterfall model, which are difficult to adapt to changing characteristics and often lead to project failures.

5.3.2 VBSQA Process Experience On A Real-World ERP Software Project Case Study

The case study was originated from a real-world project in Neusoft Co. Ltd., one of the biggest solution providers and software companies in China [Huang et. al. 2006b]. It produces software to support ERP in the fields including telecommunication, electric power, enterprise e-business, social insurance, finance, education, tax, and mobile Internet. We name all of these software products as ERP software in this dissertation.

Based on the above VBSQA process framework we conducted the initial exercise with 2 project managers from Neusoft and 2 facilitators from Nanjing University. And an undergoing ERP software project in Neusoft, which is to upgrade a Documents and Images Management System (DIMS)[1] from version 6.0 to 7.0, was used as a case study. The current 6.0 version of the DIMS software developed by Neusoft had been used in several departments of Chinese government for three years. Some departments were going to change their database platforms. At the same time, they might also add, remove or update certain attributes in the DIMS 6.0 database schema. In this case, DB administrators needed to export all the data from the old databases and import them to the upgraded ones. With the common requirements of the DB administrators from various departments, Neusoft decided to upgrade the DIMS from 6.0 to 7.0 by adding a new capability of data migration. The objective of

[1] The DIMS project is anonymous for the sake of commercial confidentiality.

this exercise was to tailor the VBSQA process to the ERP software development activities in China.

5.3.2.1 The New Process Learning Curve

Two project managers from Neusoft were given two-week series of tutorials on the VBSQA process framework and the WinWin Spiral model. Then we started to evaluate the effectiveness of the tutorials. Two project managers were to utilize the VBSQA process framework as a guideline to generate a value-based process instance based on the current ERP software development activities in order to achieve the WinWin balanced DIMS quality requirements from various project success-critical stakeholders. They developed a process instance composed of 22 ERP software development activities. During the discussion after the exercise, we detected 6 misplaced activities due to the misinterpretation of the process steps in the VBSQA process framework. We also identified 4 missing activities which should have been included in the process instance due to the misunderstanding of value-based approach and WinWin Spiral model.

This exercise and previous experience suggested that project managers typically had very short attention spans and low tolerance for new “methods” since they were usually very busy. They all suggested that we provide an easy-to-use process framework with a short learning curve.

Furthermore, project managers tended to use the VBSQA process as a guideline. They expected it to be able to adapt to the changes in the ERP software development activities and workflows. Thus, customization of VBSQA process framework toward specific software development, such as ERP software development, would improve its application and provide considerable value for project managers and software companies.

5.3.2.2 Maintaining the Flexibility of the Process

The VBSQA process framework covers all the phases and milestones in the entire software development life cycle of the WinWin Spiral model. It also includes various software development activities to incorporate the value-based consideration.

On the one hand, for most Chinese ERP solution providers, different software quality assessment criteria are set based on different business cases so that different activities may be selected to meet them. Three different process patterns (*deadline-driven, product-driven and market-trend driven*) are usually applied in the software development based on different business cases. Deadline-driven business case applies when rapidly accommodating a few minor product upgrading requirements from one or two departments within an organization. Product-driven business case applies when accommodating a request to upgrade to the next version due to the aggregation of some common upgrading requirements from various departments. In this case, the quality of the upgraded product is the process driver rather than meeting a schedule. Market-driven business case applies when the upgrade of the product is driven by the market trend or rivals' products, for instance, a change from the Client/Server architecture to the Web-based architecture. In this case, providing superior capabilities to capture greater market share as early as possible is the key process driver. To meet the different requirements of different business cases, a flexible process generation platform should be created to enable the trim and/or addition of the steps/activities based on the VBSQA process framework.

5.3.2.3 Identifying Flaws in a Process Instance

Maintaining the flexibility of the VBSQA process framework might introduce flaws during process instance creation. While a process step and/or a software development activity in the process instance could be included or excluded by project managers, the created process instance might contain flaws and/or risks. For instance, the dropped activities might cause the violation of the critical path activity dependencies so that the precondition(s) of a specific activity could not be satisfied prior to its execution.

5.3.2.4 Tradeoffs among Conflicting Quality (Q-) Attributes

Software quality is an integrative concept consisting of a number of attribute dimensions such as availability, security, performance, evolvability, schedule and cost. Thus, achieving software quality is a multi-attribute decision problem. Each project success-critical stakeholder can define his/her expected and desirable levels for each Q-attribute. The WinWin-balanced quality requirements are the reassessment and compromise of the Q-attributes among success-critical stakeholders.

As reported by the project managers, there usually existed at least one pair of conflicting Q-attributes in many ERP software projects. In the DIMS upgrade project, Performance and Evolvability were a pair of conflicting Q-attributes. There were several architecture options to select from. Direct copy between DBs favored Performance in terms of both throughput and storage space at the cost of Evolvability. XML-based architecture favored Evolvability by accommodating future changes of DB platforms and schema at the cost of Performance. Developers had to select one feasible architecture from several options in order to balance the stakeholders' conflicting Q-attribute requirements. If we could identify the conflicting Q-attributes as early as possible in the software development life cycle, we would be able to mitigate the risk of project failure by performing tradeoff analysis among conflicting Q-attributes. Our exercise showed that the stakeholder interaction activities (e.g., *External Prototype Evaluation, Architecture options external review, Selected architecture external review*) in the software development process were more effective in identifying the conflicting Q-attributes and that the stakeholder negotiation activities were more effective in performing tradeoff analysis.

5.3.2.5 The Importance of Determining Project Stakeholders' Perspectives and Interaction Point(s)

Our exercise showed that not all the stakeholders were required to have the same level (intensity) or type of involvement [West 2004] in every activity in the software development process. Thus planning the level of involvement of each stakeholder was critical.

In the mean time, the project success-critical stakeholders' interaction activities could either mitigate the software quality risks or drive the changes of stakeholders' value-propositions. And the costs (i.e., activity cost, potential rework cost) are also associated and/or resulted from such activities. Thus, planning the activities for stakeholders' interaction and negotiation at different phases of the software development life cycle might result in different Return-On-Investment (ROI). For instance, the ROI of the *External Prototype Evaluation* activity and *Architecture Options External Review* in LCO stage might be different than the counterparts in LCA stage. Therefore, determining the time of the stakeholders' interaction activities in a process instance was very important.

5.3.3 Modeling VBSQA Process Using Object Petri Nets (OPN)

To tackle the problems encountered in our first attempt at the VBSQA process training and exercise, we built a VBSQA process simulation tool VBSQA Process Generator which could be used for ERP software development in China. Some related works on process simulation have been investigated in [Kellner 1999, Padberg 2003]. The purposes of process simulation modeling are discussed in [Kellner 1999]. And a discrete-time process simulator to support software project managers in task scheduling is presented in [Padberg 2003].

The overall structure of VBSQA Process Generator is shown in Figure 5.5 [Huang et. al. 2006b]. It is composed of three components: VBSQA Process Creator, VBSQA Process Checker and VBSQA Process Simulator. The simulation results can be utilized as a feedback to adjust and improve the current VBSQA process instance. Their application is illustrated using DIMS upgrade case study in section 5.4. It aims to help industrial practitioners visualize the process and generate an appropriate and optimized VBSQA process instance based on a certain project business case.

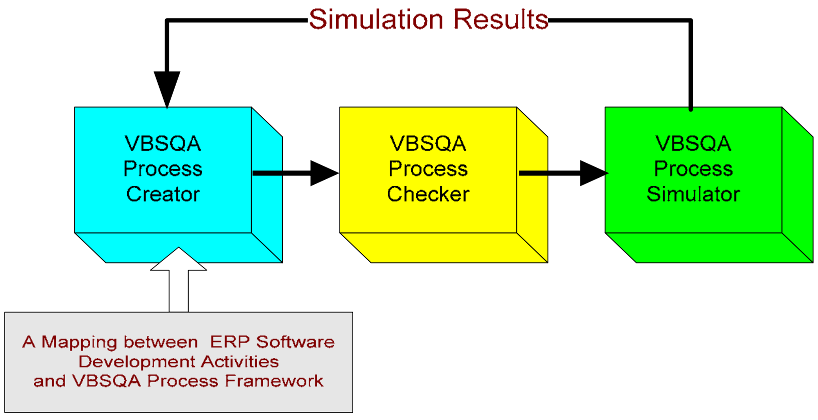


Figure 5.5. The Overall Structure of VBSQA Process Generator

In order to build the process simulation tool, first we needed to model the VBSQA process using a process language that could capture its characteristics. Furthermore, the process language should be precise enough to eventually support verification and drive simulations which could help us address some problems identified in section 5.3.2.

5.3.3.1 Purpose of VBSQA-OPN Process Modeling

Value-based software development processes tend to be stakeholder–involved with a great deal of concurrency and backtracking. VBSQA process is one of these processes with the emphasis on achieving stakeholder WinWin-balanced software quality requirements. Thus, it usually involves multiple stakeholders with different value propositions on Q-attributes and different perspectives about the on-going process.

Object Petri Nets (OPN) [Valk 1998], which is an extension of traditional Petri Nets, was chosen to model the VBSQA process. Based on the fact that the control structures of software processes are similar to those of programming languages, Osterweil proposed the idea of "Software processes are software too" [Osterweil 1987]. Because the control structures of Petri Nets (PN) are similar in expression to programming languages, they can be used to model software processes [Deiters-Gruhn 1994]. Aalst has listed three reasons for using Petri Nets for process modeling and analysis: 1) formal semantics despite the graphical nature, 2) state-based instead of event-based, 3) abundance of analysis techniques [Aalst 1998]. Furthermore, PN has the merit of modeling concurrent process activities. As an extension of traditional PN, OPN inherits these merits of PN in process modeling.

In addition, OPN supports the separation of concerns among different stakeholders' perspectives of the process by object oriented approach. Each stakeholder's process instance can be modeled in a separate Object Net (*ON*) by inheriting the activities from the relevant process steps in the System Net (*SN*) (i.e., the VBSQA process framework). We took the "object-oriented" approach in the sense that the VBSQA process framework was modeled as a *SN* which was used as a process guideline. And each stakeholder's process instance was modeled as an object that followed the workflow of the guideline to perform the ERP software development activities. Then the interaction and negotiation among stakeholders and the synchronization between each stakeholder's *ON* and the *SN* could be defined later. Thus, OPN is able to adapt to the changes in the ERP software development activities and workflows.

VBSQA-OPN model provides a feasible solution to automation or semi-automation of the VBSQA process. Section 5.3.3.2 provides the formal definitions of our VBSQA-OPN process modeling.

5.3.3.2 Formal Definitions of VBSQA-OPN Process Modeling

This section presents the formal definitions of the *VBSQA-OPN*.

Definition 1. Object Petri Nets (OPN)

A Petri net is a 3-tuple $PN = (P, T, F)$, where *P* is a finite set of places, *T* is a finite set of transitions, $P \cap T = \phi$, $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, representing the flow relation between places and transitions. Tokens, representing the pre- and post- conditions of activating a specific transition, flow from one place to another. In the diagram, we use a circle to represent a place, use a bar to represent a transition and use a solid dot to represent a token. Please refer to [Reisig 1985] for the basic concepts of Petri nets.

An OPN is a 3-tuple $OPN = (SN, ON_s, \rho)$. This definition supports multi-objects and it is extended from the Valk's definition [Valk 1998].

- $SN = (P, T, W)$ is a Petri net, named System Net, $W \subseteq (P \times T) \cup (T \times P)$. The tokens in SN refer to the Object Nets defined below.
- $ON_S = \{ON_1, ..., ON_n\}$ (*n*>1) is a finite set of Object Nets in OPN , $ON_i = (B_i, E_i, F_i)$ is a Petri net, named Object Net. $F_i \subseteq (B_i \times E_i) \cup (E_i \times B_i)$.
- SN and ON_s synchronize via "channels" (ρ). ρ is the synchronous relation between SN and ON_s , $\rho \subseteq T \times E$, where $E := \bigcup \{E_i \mid 1 \le i \le n\}$.
- To support multi-objects in VBSQA-OPN model, we extend a special Occurrence Rules based on the Valk's Three Occurrence Rules [Valk 1998], which are omitted here.

Definition 2. VBSQA-OPN

$VBSQA-OPN(SN, ON_s, \rho)$ is the modeling of VBSQA process based upon OPN, where

- $SN = (P, T, W)$ is the VBSQA process framework. Here, the transition set T represents the steps/milestones and it is divided into two disjoint subsets T_{syn} and T_{st}, where $T = T_{syn} \cup T_{st}$.
    - ➢ T_{syn} is a set of synchronous transitions, which represent the process steps/milestones that are actually performed by stakeholder(s) in their process instances.
    - ➢ T_{st} is a set of status transitions. A status transition can only immediately follow a synchronous transition in the SN.

The graphical representations of the two types of transitions are shown in the legend of Figure 5.6. The tokens in SN refer to Object Nets (i.e., stakeholders' process instances) and point to the marking of Object Nets.

Here, SN can be either the entire VBSQA process framework SN_0 or a tailoring from SN_0 based on the project business cases. Given the set of transitions to be deleted, we can construct the tailored process framework SN as follows.

The set of transitions to be deleted are denoted as DEL. When transition t is deleted,

1) Let $DEL := \{t\}$,

2) $DEL := DEL \cup \{$the status transitions immediately following $t\}$

3) $\forall x \in DEL,$ if $\exists i, x \in Loop_i$ then $DEL = DEL \cup Loop_i$. $Loop_i$ denotes the set of transitions which belong to the same closed loop in the directed graph.

4) Delete the transitions in DEL and add the arc(s) from the places before DEL to the transitions following DEL.

- $ON_S = \{ON_1, ..., ON_n\}$ (n>1) represents a set of process instances of stakeholders and $ON_i = (B_i, E_i, F_i)$ is the process instance of stakeholder *i*. In ON_i, the transition set E_i represents the ERP software development activities that should be performed by stakeholder *i* and it includes three disjoint subsets, E_{iauto}, E_{isyn}, E_{ist}, where $E_i = E_{iauto} \cup E_{isyn} \cup E_{ist}$.
  - ➢ E_{iauto} is a set of the object-autonomous transitions, which represents the autonomous activities of stakeholder *i* that can not be mapped to any given step/milestone in SN (i.e., VBSQA process framework).
  - ➢ E_{isyn} is a set of synchronous transitions, which can be mapped to the steps/milestones in SN (i.e., VBSQA process framework) and has synchronous relation with SN.
  - ➢ E_{ist} is a set of status transitions, which can only immediately follow a synchronous transition.

The graphical representations of the three types of transitions are shown in the legend of Figure 5.7. The tokens in ON_i represent the Q-attributes (e.g., Performance, Evolvability, Schedule, Cost, etc.) concerned by the stakeholder *i*.

- $\rho = T_{syn} \times E_{syn}$ $(E_{syn} := \bigcup\{E_{isyn} \mid 1 \le i \le n\})$ defines the synchronous relation between SN and ON_s, that is, a mapping between the VBSQA process framework steps/milestones and ERP software development activities.
- Guard functions are defined to set the activation condition(s) for some transitions. In this case, a transition $t \in T$ is activated in a marking M (denoted as $M[t>)$ iff $M \ge \bullet t$ and the t transition's guard functions are both satisfied.
- **Constraint 1**. The chronological order of ERP software development activities in the stakeholders' process instances ON_s is consistent with the chronological order of VBSQA process framework steps/milestones in the SN based on their mapping. For a $VBSQA-OPN=(SN, ON_s, \rho)$, if there is a path from step A_s to B_s in the SN, denoted as $A_s \prec B_s$, and in the ON_s, there exist two ERP activities A_o and B_o such that $(A_s, A_o) \in \rho$ and $(B_s, B_o) \in \rho$, then there must exist a path from A_o to B_o, denoted as $A_o \prec B_o$.
- **Constraint 2**. Critical Path Activity Dependency. For a $VBSQA-OPN=(SN, ON_s, \rho)$, if transition A_s must be completed before transition B_s (i.e. $A_s \prec B_s$) in the SN, (denoted as $B_s \mapsto A_s$), and transition B_o exists in ON_i, that is, $\exists B_o \in E_{i,}$ $(B_s, B_o) \in \rho$ in $ON_i = (B_i, E_i, F_i)$, then $\exists A_o \in E_j, (A_s, A_o) \in \rho$ in $ON_j = (B_j, E_j, F_j)$ (denoted as $B_o \mapsto A_o$).

5.3.4 Application of VBSQA Process Generator Built on VBSQA-OPN Model

Based on the VBSQA-OPN Modeling of the VBSQA process, the VBSQA Process Generator was built. We asked two project managers to apply this tool on the DIMS upgrade case study. Section 5.3.4.1 shows how to use the VBSQA Process Creator to create the process instances for project success-critical stakeholders based on the VBSQA process framework. Section 5.3.4.2 illustrates how to identify the flaws of a process instance based on defined process constrains in the VBSQA Process Checker. Section 5.3.4.3 presents some simulation results of the ERP VBSQA process.

5.3.4.1 VBSQA Process Creator: Creating an ERP VBSQA Process Instance

5.3.4.1.1 Mapping the ERP Software Development Activities into VBSQA Process Framework

To shorten the VBSQA process learning curve and to reduce the flaws such as the misplacement of ERP development activities when creating a process instance, we mapped the ERP software development activities into each step/milestone in the VBSQA process framework. Table 5.11 shows a part of this mapping based on the current ERP software development activities. In the VBSQA-OPN model, VBSQA process framework was modeled as the System Net (*SN*) and each stakeholder class's process instance was modeled as an Object Net (*ON*) inherited from the *SN*. Note that we only distinguished different stakeholder classes in creating a process instance but not the various roles in one stakeholder class. For instance, we assumed that IV&V team and testing team belong to the Developers. Thus, to create a process instance for a stakeholder, project managers just needed to select a specific activity mapped into the VBSQA process step and added it into the plan for this stakeholder. The chronological orders of these activities were automatically inherited from the *SN*, which eliminated the process flaws of misplaced activities due to the misinterpretation of the process steps in the VBSQA process framework as discussed in section 3.3.2. And new activities which were not mapped into any step/milestone could be added into the stakeholders' *ON* as needed. Furthermore, if the ERP software development activities and/or workflows are changed in the future, we will only need to change the mapping.

VBSQA Process Framework Steps/Milestones (System Net)	**ERP Software Development Activities (Object Nets)**
Initiate project	Acquire system upgrade requirements (Developer)
Project cost/benefit analysis	Estimate system upgrade cost & develop DMR results chain (Developer)
	Verify system upgrade cost (System Acquirer)
SCS define acceptable & desired values for Q-attributes	Requirement elicitation meeting
	Groupware WinWin negotiation
Risk analysis & architecture/technology evaluation	Internal prototype evaluation (Developer)
	External prototype evaluation
Identify conflicting Q-attributes & perform tradeoff analysis	Identify conflicting Q-attributes & perform tradeoff analysis
SCS adjust acceptable values for Q-attributes	Stakeholder renegotiation
System top-level design and initial Feasibility Rationale Description (FRD)	System top-level design (Developer)
LCO Review	Architecture options internal review (Developer)
	Architecture options external review
SCS refine acceptable & desired values for Q-attributes	Requirement elicitation meeting
	Groupware WinWin negotiation
System detailed design and detailed Feasibility Rationale Description (FRD)	System detailed design (Developer)
LCA Review	Selected architecture internal review (Developer)
	Selected architecture external review
Core capability implementation	Core capability implementation (Developer)
Value-based core capability testing	Internal core capability testing (Developer)
CCD	Internal core capability demo (Developer)
	Onsite core capability demo
Remaining features implementation	Complete system implementation (Developer)
IOC Acceptance Review	Onsite System Acceptance Review

Table 5.11. Mapping the ERP software development activities into VBSQA process framework steps/milestones

5.3.4.1.2 DIMS Upgrade Case Study: Creating the ERP VBSQA Process Instance

In the DIMS upgrade case study, we identified 4 stakeholder classes including *System Acquirer, DB Administrators, Software Maintainers* and *Developers*. Figure 5.6 shows a segment of the *SN* (i.e, VBSQA process framework). Figure 5.7 illustrates the corresponding segment of the *ON* representing a process instance for the Developers generated from the *SN*. Figure 5.8 illustrates the corresponding segment of the *ON* representing a process instance for the System Acquirer. Figure 5.9 shows an example of the creation of the Developer process instance using VBSQA Process Creator.

When the mouse cursor was rested over a particular process step/milestone of the *SN* in the VBSQA Process Creator as shown in Figure 5.9, the applicable procedure/approach, if any, was displayed in a textbox. In this way, a project manager could associate the procedure/approach to the specific activity mapped to this process step/milestone. Similarly, when the mouse cursor was rested over a particular activity in the *ON* representing the stakeholder process instance, the corresponding stakeholder responsibilities (e.g., the documents and/or product to be delivered) were displayed in a textbox. Therefore, by creating different process instances for various stakeholders, we could separate one stakeholder's responsibilities from others' with respect to the activities that he/she was involved in.

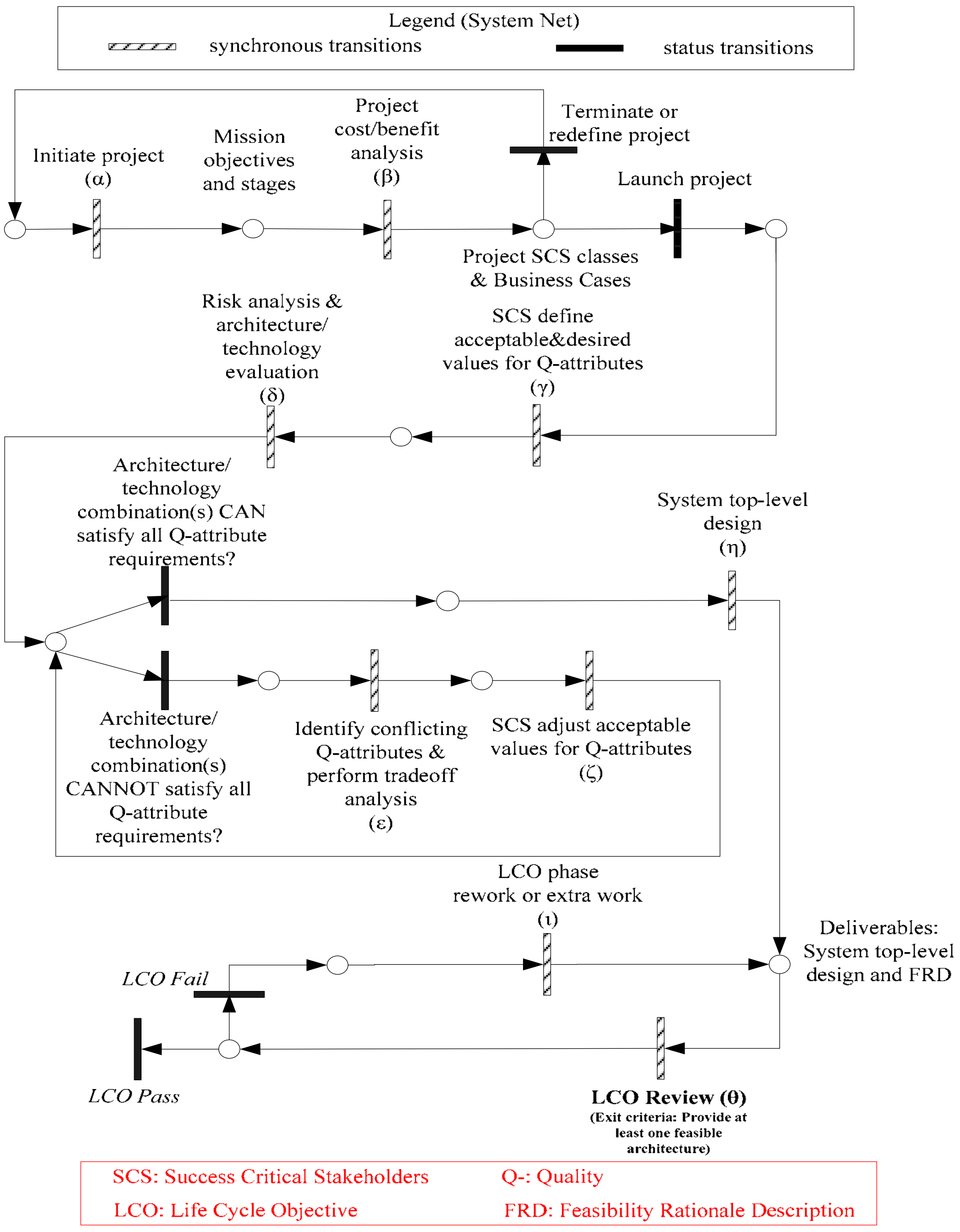

Legend (System Net)
synchronous transitions
status transitions
Initiate project (α)
Mission objectives and stages
Project cost/benefit analysis (β)
Terminate or redefine project
Launch project
Project SCS classes & Business Cases
Risk analysis & architecture/ technology evaluation (δ)
SCS define acceptable&desired values for Q-attributes (γ)
Architecture/ technology combination(s) CAN satisfy all Q-attribute requirements?
System top-level design (η)
Architecture/ technology combination(s) CANNOT satisfy all Q-attribute requirements?
Identify conflicting Q-attributes & perform tradeoff analysis (ε)
SCS adjust acceptable values for Q-attributes (ζ)
LCO phase rework or extra work (ι)
Deliverables: System top-level design and FRD
LCO Fail
LCO Pass
LCO Review (θ)
(Exit criteria: Provide at least one feasible architecture)
SCS: Success Critical Stakeholders
Q-: Quality
LCO: Life Cycle Objective
FRD: Feasibility Rationale Description

Figure 5.6. VBSQA-OPN System Net (*SN*): the LCO Phase of VBSQA Process Framework

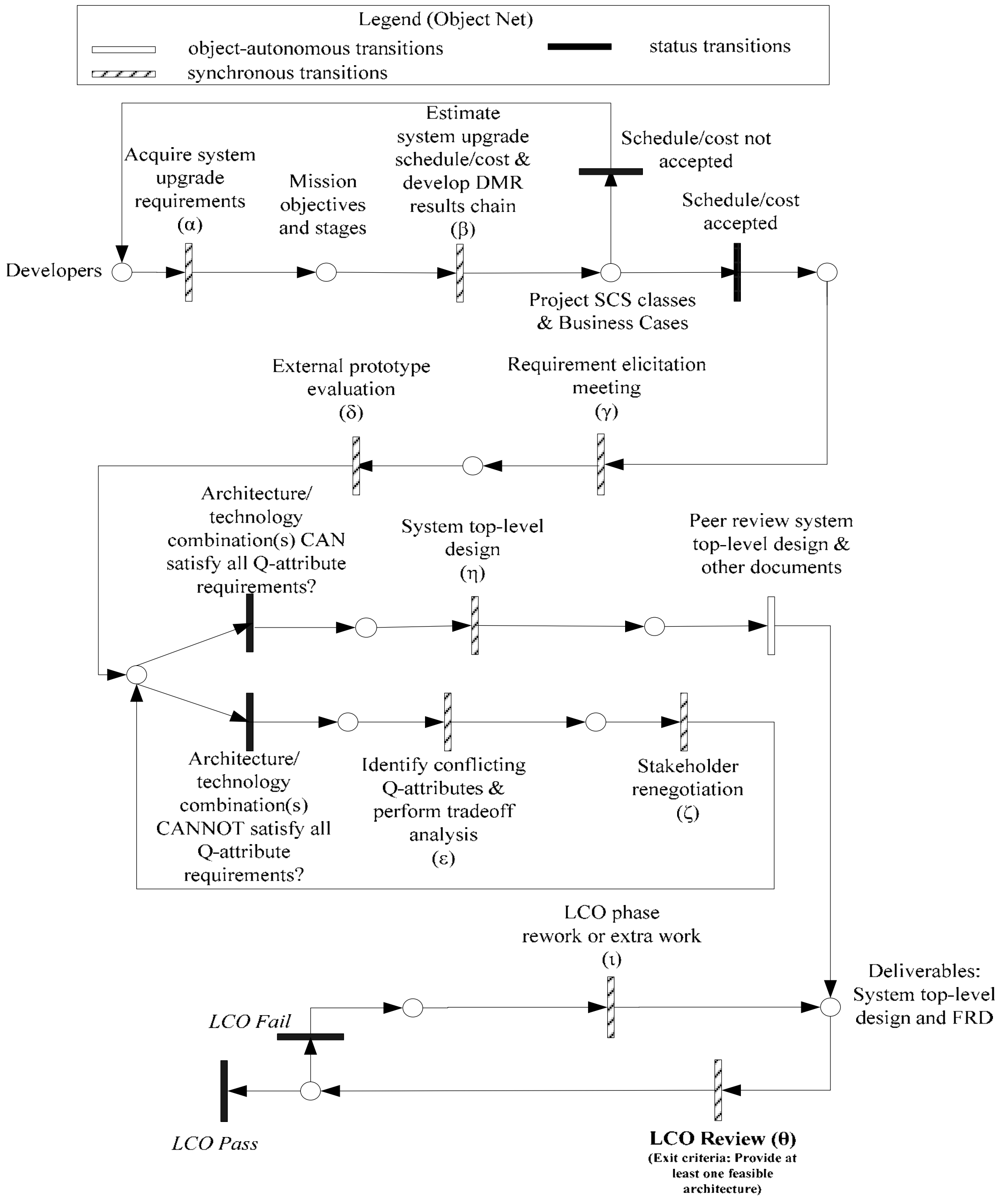

Legend (Object Net)
object-autonomous transitions
synchronous transitions
status transitions
Acquire system upgrade requirements (α)
Mission objectives and stages
Estimate system upgrade schedule/cost & develop DMR results chain (β)
Schedule/cost not accepted
Schedule/cost accepted
Developers
Project SCS classes & Business Cases
External prototype evaluation (δ)
Requirement elicitation meeting (γ)
Architecture/ technology combination(s) CAN satisfy all Q-attribute requirements?
System top-level design (η)
Peer review system top-level design & other documents
Architecture/ technology combination(s) CANNOT satisfy all Q-attribute requirements?
Identify conflicting Q-attributes & perform tradeoff analysis (ε)
Stakeholder renegotiation (ζ)
LCO phase rework or extra work (ι)
Deliverables: System top-level design and FRD
LCO Fail
LCO Pass
LCO Review (θ)
(Exit criteria: Provide at least one feasible architecture)

Figure 5.7. VBSQA-OPN Developer Object Net (*ON*): the LCO Phase of the Developer Process Instance Generated from the SN

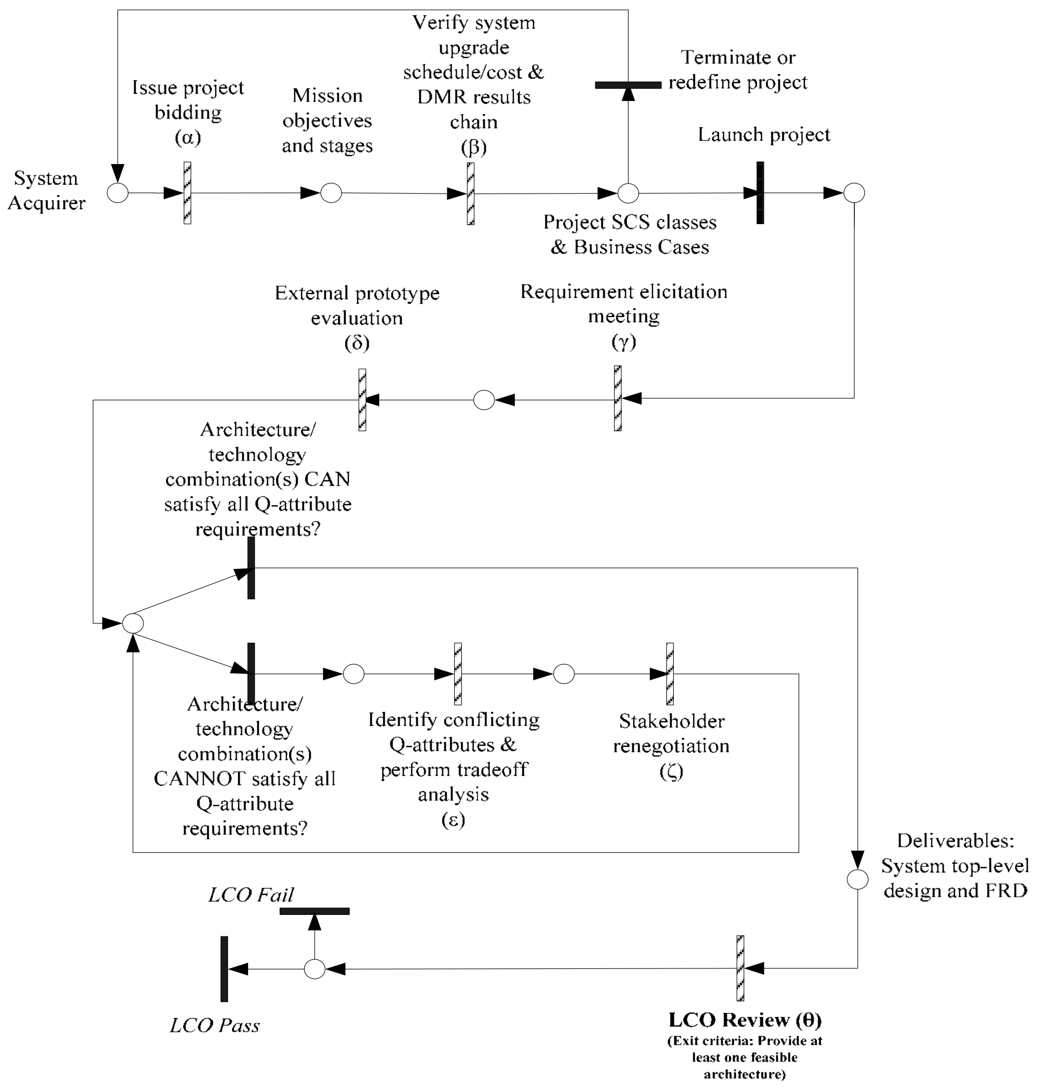
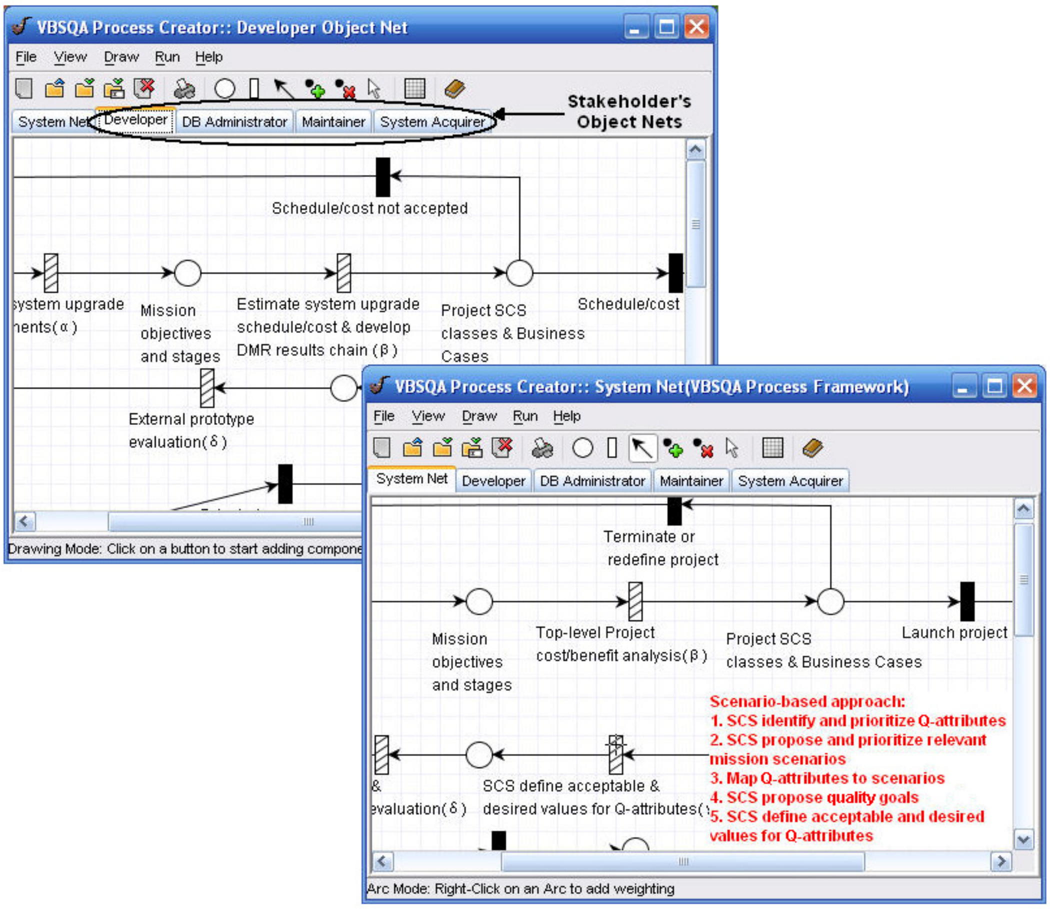

Verify system upgrade schedule/cost & DMR results chain (β)
Terminate or redefine project
Issue project bidding (α)
Mission objectives and stages
Launch project
System Acquirer
Project SCS classes & Business Cases
External prototype evaluation (δ)
Requirement elicitation meeting (γ)
Architecture/ technology combination(s) CAN satisfy all Q-attribute requirements?
Architecture/ technology combination(s) CANNOT satisfy all Q-attribute requirements?
Identify conflicting Q-attributes & perform tradeoff analysis (ε)
Stakeholder renegotiation (ζ)
Deliverables: System top-level design and FRD
LCO Fail
LCO Pass
LCO Review (θ)
(Exit criteria: Provide at least one feasible architecture)

Figure 5.8. VBSQA-OPN System Acquirer Object Net (*ON*): the LCO Phase of the System Acquirer Process Instance Generated from the SN


VBSQA Process Creator:: Developer Object Net
File View Draw Run Help
System Net Developer DB Administrator Maintainer System Acquirer
Stakeholder's Object Nets
Schedule/cost not accepted
ystem upgrade
ents(α)
Mission objectives and stages
Estimate system upgrade schedule/cost & develop DMR results chain (β)
Project SCS classes & Business Cases
Schedule/cost
External prototype evaluation(δ)
Drawing Mode: Click on a button to start adding compone
VBSQA Process Creator:: System Net(VBSQA Process Framework)
File View Draw Run Help
System Net Developer DB Administrator Maintainer System Acquirer
Terminate or redefine project
Mission objectives and stages
Top-level Project cost/benefit analysis(β)
Project SCS classes & Business Cases
Launch project
&
evaluation(δ)
SCS define acceptable & desired values for Q-attributes(
Arc Mode: Right-Click on an Arc to add weighting

Scenario-based approach:
1. SCS identify and prioritize Q-attributes
2. SCS propose and prioritize relevant mission scenarios
3. Map Q-attributes to scenarios
4. SCS propose quality goals
5. SCS define acceptable and desired values for Q-attributes

Figure 5.9. VBSQA Process Creator: VBSQA Process Framework (System Net) and the Generated Developer Process Instance (Object Net)

5.3.4.2 VBSQA Process Checker: Identifying the Flaws in a VBSQA Process Instance

Based on the project business case, the project manager could choose to skip some steps in the VBSQA process framework during the ERP VBSQA process instance creation for success-critical stakeholders. That is, it allowed project managers to inherit a *NULL* activity from each step in the *SN* (i.e., VBSQA process framework). However, such flexibility provided by the tool could be both a strength and a weakness. It might introduce the flaws of missing activities which could cause the violation of critical path activity dependencies in a process instance.

One way to validate the process was to provide a process analysis capability to verify that critical path activity dependency constraints were not violated by the process definition. These constraints were represented as formal properties defined in the VBSQA-OPN System Net (*SN*) and implemented in the VBSQA Process Checker. Some examples of the activity dependency constraints in the *SN* could be as follows:

- *SCS define acceptable & desired values for Q-attributes* must be completed before *Risk analysis & architecture/technology evaluation*
- *Risk analysis & architecture/technology evaluation* must be completed before *System top-level design*
- *System top-level design* must be completed before *LCO Review*
- *Value-based core capability testing* must be completed before *CCD*
- And they needed to be translated into the precise formal definitions based on the Constraint 2 of Definition 2 in section 5.3.3.2.

For instance, the activity *System top-level design* had been planned in the LCO phase of the Developers' process instance. However, neither *Internal prototype evaluation* nor *External prototype evaluation* which were mapped to the *Risk analysis & architecture/technology evaluation* as shown in Table 5.11 was planned in any stakeholder's process instance. After analyzing the stakeholders' process instances in ON_s based on the defined critical path activity dependency constraints, the VBSQA Process Checker would display a warning message as "*Risk analysis & architecture/technology evaluation* must be completed before *System top-level design*".

5.3.4.3 VBSQA Process Simulator: Comparing the ROI of Synchronous Stakeholder Interaction Activities

The synchronous stakeholder interaction activities in the VBSQA process were usually effective in mitigating the software quality risks. At the same time, higher activity costs and different potential rework costs were incurred by such activities in different phases of software development. Thus, performing the stakeholder interaction activity at different phases of the software development life cycle might result in different Return-On-Investment (ROI). However, there lacked of the quantitative analysis approaches and simulation tools to help project managers determine when to perform the synchronous stakeholder interaction activities during software development.

In this section, we present the simulation results of different ROI for various stakeholder interaction activities in different software development phases. The ROI was computed as following:

$$ROI = \frac{Value - Cost}{Cost} \qquad (5.2)$$

5.3.4.3.1 Value Earned: the Synchronous Stakeholder Interaction Activity vs. Developer Internal Activity

Let's assume that totally *m* Q-attributes were identified for the project. And *n* (n=4) software development phases were defined in the VBSQA process framework (LCO, LCA, CCD and IOC).

$$Value = \sum_{i=1}^{m} ((1 - \prod_{j=1}^{n} (1 - E_{ij})) \times R_i \times V) \tag{5.3}$$

V: the <u>total value of the project</u>;
***E_{ij}* (0-1)**: the effectiveness of a specific process activity on mitigating the risk of Q-attribute *i* if it is performed in phase *j*;
***R_i* (0-1)**: the risk of Q-attribute *i* to the <u>total value of the project</u>.

In DIMS upgrade case study, two project managers provided the inputs for each parameter based on their experience and expert judgement. <u>Total value of the project</u> was estimated as the contracted payment ($50,000) that Neusoft would obtain from the system acquirer after the project was successfully completed by satisfying of all success-critical stakeholders' Q-attribute requirements. Totally 4 Q-attributes and their related operational scenarios were identified for this project as shown in Table 5.12.

Q-attributes	**Related Scenario**	**Priority**	***R_i* (0-1)**
Performance	Complete data migration from the old DB to the upgraded DB within 1 day and within required storage space	High	0.8
Evolvability	Accommodate different DB platforms and schema in data migration	Medium	0.4
Schedule	—	High	—
Cost	—	High	—

Table 5.12. DIMS Upgrade Project: Q-attributes and Their Risks to the Project Value (*R_i*)

We treated the failure of achieving a Q-attribute requirement as the risk to the total value of the project. Table 5.12 lists the risk of Performance and Evolvability to the project value, denoted as R_i. R_i is the product of the risk impact and the probability of risk occurrence. We defined the "risk impact" as *the proportion of the total project value that would have been lost if that Q-attribute risk had occurred*. Thus R_i was from 0 to 1. In this project, Performance was a High priority requirement from DB administrators and failure of achieving it would result in 80% loss of the project value. Evolvability was a Medium priority requirement from software maintainer and failure of achieving it would result in 40% loss of the project value. In this case study, we did not take into account the other two Q-attributes (Schedule, Cost) because they were not the major risks to this project.

Some steps in the VBSQA process framework such as Risk analysis & architecture/technology evaluation, LCO Review, LCA Review, CCD aimed to mitigate the quality risks. As shown in Table 5.11, two types of ERP software development activities were mapped to each of them, which were developer internal activities and synchronous stakeholder interaction activities. The examples of the former were *Internal prototype evaluation*, *Architecture options internal review*, *Selected architecture internal review*, *Internal core capability demo* as shown in the right column of Table 5.11. These activities were accomplished only by developers without other stakeholders' participation. They were usually less effective in identifying Q-attribute risks. The examples of the latter were *External prototype evaluation*, *Architecture options external review*, *Selected architecture external review*, *Onsite core capability demo*. These activities were accomplished by developers and other success-critical stakeholder(s). Stakeholders were able to evaluate the prototype(s), review the architecture(s) or test the core capabilities together under the realistic operational environment. Thus, these activities were usually more effective in identifying Q-attribute risks. Furthermore, these activities at different phases of the software development life cycle also had different effectiveness in identifying Q-attribute risks. Table 5.13 shows the effectiveness of a specific process activity on mitigating the risk of Q-attribute *i* if it was performed in phase *j*, denoted as E_{ij}. Project managers provided the estimate of E_{ij} as the *proportion by which the risk of Q-attribute i would have been reduced if that process activity had been performed in phase j*. Thus E_{ij} was from 0 to1. Note that we treated the risk mitigation E_{ij} in its most general sense, which incorporated both the decrease of the probability of risk occurrence and their impact on the project value.

Project Phases	**Process Activities**	**Risk Mitigation (E_{ij}) (0-1)**	
		Performance	**Evolvability**
LCO (Life Cycle Objectives)	Architecture options internal review	0.2	0.2
	Architecture options external review	0.6	0.6
LCA (Life Cycle Architecture)	Selected architecture internal review	0.3	0.3
	Selected architecture external review	0.8	0.8
CCD (Core Capability Demo)	Internal core capability demo	0.2	0.2
	Onsite core capability demo	0.5	0.5
IOC (Initial Operational Capability)	Onsite system acceptance review	0.3	0.3

Table 5.13. DIMS Upgrade Project: the Effectiveness of Developer Internal Activities vs. Stakeholder Interaction Activities on Q-attribute Risk Mitigation (E_{ij})

5.3.4.3.2 Cost: the Synchronous Stakeholder Interaction Activity vs. Developer Internal Activity

Two types of cost, the activity cost and the potential rework cost, were associated with a synchronous stakeholder interaction activity or a developer internal activity. The cost was computed as following:

$$Cost = \sum_{j=1}^{n} C_{aj} + C_r \tag{5.4}$$

C_{aj}: the cost of a process activity at phase *j*;
C_r: the potential rework cost.

5.3.4.3.2.1 ***Activity Cost (C_{aj})***

The cost of the developer internal activity in DIMS upgrade project was estimated as $500 by ERP software project managers. The synchronous stakeholder interaction activity usually had 2 or 3 time higher activity cost, estimated as $1,500.

5.3.4.3.2.2 ***Rework Cost (Cr)***

Whenever a Q-attribute risk was identified by a process activity, some amount of rework was needed as a remedy. Table 5.14 shows the potential rework cost C_r at 4 phases of VBSQA software development process. In the best case, rework was only needed for the current phase. However, sometimes rework extended to the previous phases. In the worst case, the rework needed to be done from the beginning of the project. The numbers in Table 5.14 show the rework cost C_r from phase(S) to phase(F). The numbers in the diagonal of Table 5.14 represent the rework cost within the LCO, LCA, CCD and IOC phases respectively. For instance, the cost of only reworking LCA phase was $9,000, the cost of reworking LCA and

CCD phases was \$35,000 and the cost of reworking LCA, CCD and IOC phases was \$46,000. Note that if developers needed to rework both the LCA and CCD phases because a risk was identified at the *Onsite Core Capability Demo*, the rework cost provided by project managers was \$35,000, which was larger than the sum of the rework cost within LCA and CCD phases (\$9,000 + \$19,000). They explained that since developers had to change the detailed architecture design and to redo the *Core Capability Implementation*, they usually needed extra effort to become familiar with the programming techniques for the new architecture design. Based on the ERP project managers, developer internal activities usually incurred little rework.

Rework Cost C_r (\$)		**Phase (F)**			
		LCO	**LCA**	**CCD**	**IOC**
Phase (S)	**LCO**	3,000	12,000	38,000	50,000
	LCA	—	9,000	35,000	46,000
	CCD	—	—	19,000	31,000
	IOC	—	—	—	11,000

Table 5.14. DIMS Upgrade Project: Potential Rework Cost C_r at Different Phases of VBSQA Software Development Process

5.3.4.3.2.3 <u>Simulation Results: ROI</u>

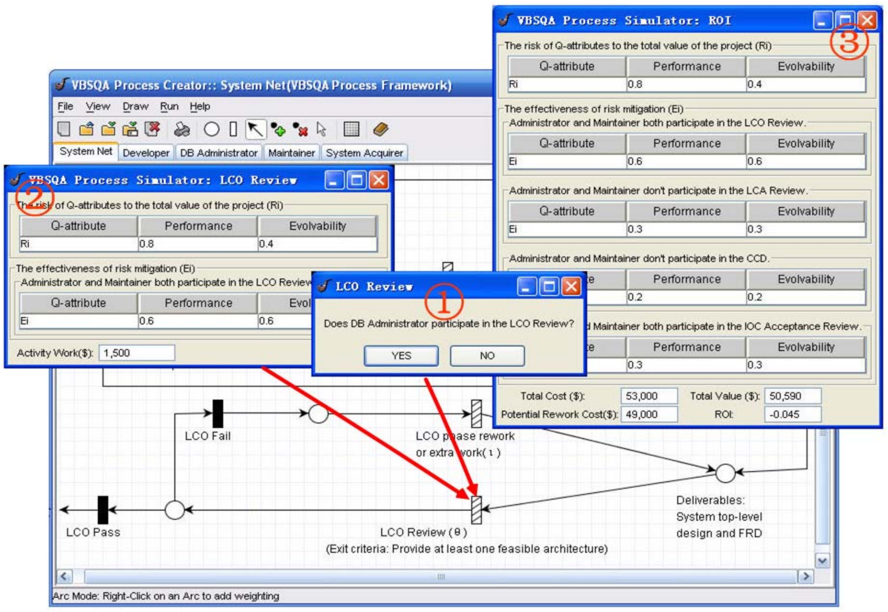
Assuming that the synchronous stakeholder interaction activity (i.e., *Onsite System Acceptance Review*) was required in the IOC phase, we enumerated the possible combinations of stakeholder interaction activities and developer internal activities in the first three phases of software development life cycle (LCO, LCA and CCD). As shown in the second column of Table 5.15, LCO(i) denotes that we performed the developer internal activity (i.e., *Architecture Options Internal Review*) in the LCO phase. LCO(s) denotes that we performed synchronous stakeholder interaction activities (i.e., *Architecture Options External Review*) in the LCO phase. The same notation applies for other phases.

Given the inputs from two project managers, our simulation computed two ROI's for each process activity combination except the first one as shown in Table 5.15. One was for the worst-case scenario; the other was for the best-case scenario in terms of the potential rework cost. In the worst-case scenario, we assumed that rework happened after each synchronous stakeholder interaction activity. And performing such activity in a certain phase could only avoid the future rework extending to this phase. For instance, only performing the *Selected Architecture External Review* in the LCA phase (i.e., LCO(i)\LCA(s)\CCD(i)\IOC(s)) would incur the rework on both LCO and LCA and another rework on both CCD and IOC. In the best-case scenario, we assumed that once we performed such activity in a certain phase the rework would only be needed from the beginning of the project to this phase and it could avoid all the future rework incurred by the Q-attribute risks afterwards.

Even in the worse case scenario, S*elected Architecture External Review* in the LCA phase was particularly effective in improving the ROI since all the combinations with this activity (see 3, 5, 7, 8) in Table 5.15 produced relatively high worst-case ROI (0.162, 0.194, 0.205, 0.215). In the best-case scenario, *Architecture Options External Review* in the LCO phase was particularly effective in improving the ROI because we assumed that it avoided all the future rework incurred by Q-attribute risks after the LCO phase. Both results implied that performing synchronous stakeholder interaction activities in the architecting phase of a software project could produce higher return in terms of software quality risk mitigation. Figure 5.10 is a snapshot of using VBSQA Process Simulator to compute the ROI of a combination of stakeholder interaction activities and developer internal activities based on the System Net (i.e., VBSQA process framework).

	Process Activity Combinations	ROI	
		Worst-Case Scenario	Best -Case Scenario
1	LCO(i)\ LCA(i) \ CCD(i) \ IOC(s)	-0.223	—
2	LCO(s)\ LCA(i) \ CCD(i) \ IOC(s)	-0.045	6.23
3	LCO(i)\ LCA(s) \ CCD(i) \ IOC(s)	0.162	2.414
4	LCO(i)\ LCA(i) \ CCD(s) \ IOC(s)	-0.09	0.149
5	LCO(s)\ LCA(s) \ CCD(i) \ IOC(s)	0.194	6.165
6	LCO(s)\ LCA(i) \ CCD(s) \ IOC(s)	0.002	5.765
7	LCO(i)\ LCA(s) \ CCD(s) \ IOC(s)	0.205	2.33
8	LCO(s)\ LCA(s) \ CCD(s) \ IOC(s)	0.215	5.48

Table 5.15. Comparing the ROI of Various Combinations of Synchronous Stakeholder Interaction Activities and Developer Internal Activities


VBSQA Process Creator:: System Net(VBSQA Process Framework)
File View Draw Run Help
System Net Developer DB Administrator Maintainer System Acquirer
①
②
③
YES NO
LCO Fail
LCO phase rework or extra work(ι)
LCO Pass
LCO Review (θ)
(Exit criteria: Provide at least one feasible architecture)
Deliverables: System top-level design and FRD
Arc Mode: Right-Click on an Arc to add weighting

VBSQA Process Simulator: LCO Review

The risk of Q-attributes to the total value of the project (Ri)

Q-attribute	Performance	Evolvability
Ri	0.8	0.4

The effectiveness of risk mitigation (Ei)

Administrator and Maintainer both participate in the LCO Review

Q-attribute	Performance	Evol
Ei	0.6	0.6

Activity Work($): 1,500

LCO Review

Does DB Administrator participate in the LCO Review?

VBSQA Process Simulator: ROI

The risk of Q-attributes to the total value of the project (Ri)

Q-attribute	Performance	Evolvability
Ri	0.8	0.4

The effectiveness of risk mitigation (Ei)

Administrator and Maintainer both participate in the LCO Review.

Q-attribute	Performance	Evolvability
Ei	0.6	0.6

Administrator and Maintainer don't participate in the LCA Review.

Q-attribute	Performance	Evolvability
Ei	0.3	0.3

Administrator and Maintainer don't participate in the CCD.

e	Performance	Evolvability
	0.2	0.2

d Maintainer both participate in the IOC Acceptance Review.

e	Performance	Evolvability
	0.3	0.3

Total Cost ($): 53,000 Total Value ($): 50,590
Potential Rework Cost($): 49,000 ROI: -0.045

Figure 5.10. VBSQA Process Simulator: Computing the ROI of a Combination of Stakeholder Interaction Activities and Developer Internal Activities in an ERP VBSQA Process

5.3.4.3.2.4 Project Results and Feedback on the ROI Simulation Results

The results in Table 5.15 enabled the project managers to rapidly assess the best-case and worst-case outcomes for their risk–mitigation decision options. Based on the feedback from two ERP project managers, the best-case scenario was usually not applicable in this project especially for the Performance attribute. In the LCO Review, developers usually could only provide the top-level system design and the non-functional prototype. Thus, the assumption that the *Architecture Options External Review* in the LCO phase could avoid all the future rework was too optimistic. However, in the LCA Review, when the detailed system design and the functional prototype were available, the assumption of avoiding the future rework after LCA phase was more applicable. Therefore, the real case scenario for Performance and Evolvability attributes in the DIMS upgrade project was closer to the worst-case scenario.

As a result of this analysis, the DIMS project managers committed to hold a *Selected Architecture External Review* at the end of the LCA phase to evaluate the performance of selected XML architecture with totally 3,840,000 DB records. With the participation of DB administrators, software maintainers and developers in this activity, they identified the architectural risk on the Performance of data migration because the entire memory would be consumed by totally 97 intermediate XML files generated. After stakeholders' renegotiation, the developers re-architected the capability as Direct Copy with additional algorithms to only accommodate certain DB platforms and schema. Based on the project managers, without such analysis results they would have planned the process activities in a value-neutral way (e.g., holding *Selected Architecture Internal Review* at the end of the LCA phase only to save some activity cost) which would have increased the chance of project failure.

5.3.5 VBSQA-OPN Modeling Costs and Benefits

The VBSQA-OPN modeling and application costs and benefits are summarized in Figure 5.11 based on the Neusoft DIMS upgrade project. The benefits are measured by the saved efforts (in hours) in terms of : 1) mutual learning; 2) developing project plan; 3) verification and validation (V&V) of project plan; 4) improving plan; 5) early vs. late plan rework.

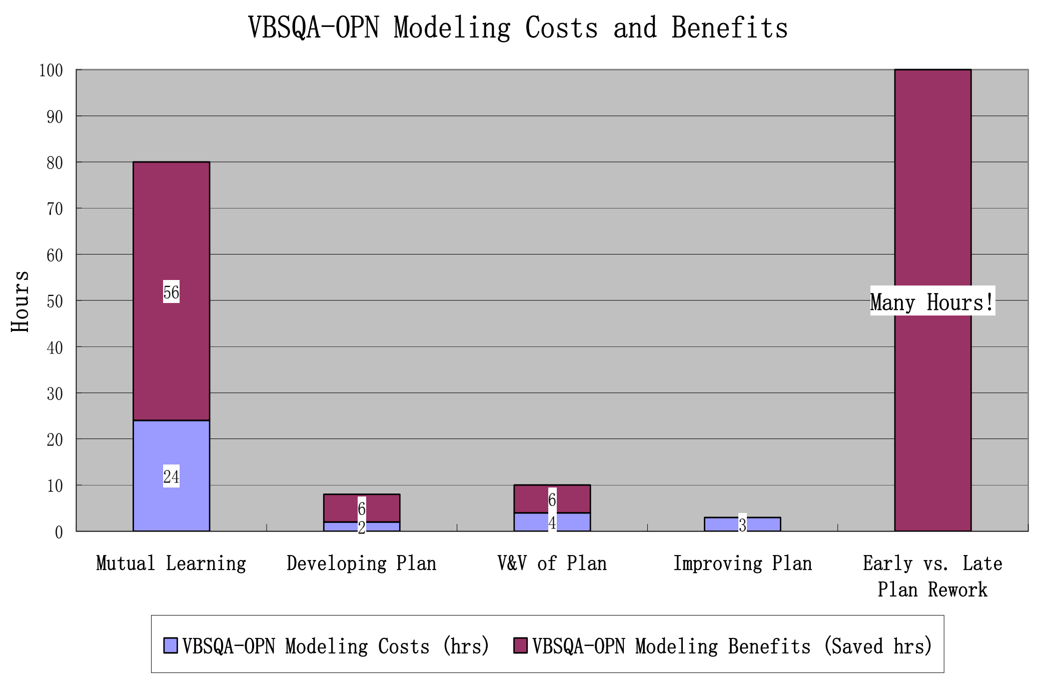


Figure 5.11. VBSQA-OPN Modeling Costs and Benefits in Neusoft DIMS Upgrade Project

In the meantime, the VBSQA-OPN modeling and VBSQA Process Generator improved user satisfaction in that

- Project managers and stakeholders became more confident in generated project plans.
- ROI simulation results were very helpful in optimizing project plans
- The VBSQA Process Generator is easy to learn.
- Neusoft DIMS upgrade project was successfully built and working to plan.

5.3.6 Tailor VBSQA Process to Project Business Cases

5.3.6.1 Characteristics of Three Business Cases in ERP Software Development

For most ERP solution providers in China, different software quality assessment criteria are set based on different business cases [Reifer 2002] so that different process strategies should be selected to meet them. Three process strategies (schedule-driven, product-driven and market-trend driven) can be selectively applied in the ERP software development based on different business cases. To improve the flexibility of the VBSQA process, the risk-based process strategy decision-making approach embedded in the VBSQA process framework can be applied to tailor the process strategy to different project business cases [Huang et. al. 2006a]. When tailoring the process, we may skip some process steps/milestones, relax the deliverables/outputs of a particular process step/milestone, select a particular ERP software development activity, or decide the participants of a process activity.

Firstly, we shall determine whether the project is dominated by schedule risks or quality risks. Table 5.16 compares the different characteristics of three typical business cases in ERP software projects. Then we will use the real-world ERP software system, a Documents and Images Management System (DIMS) developed by Neusoft, as an example to illustrate how to use the risk-based process decision-making approach to tailor the VBSQA process to three different business cases. Four success-critical stakeholder classes were identified in DIMS project, including the System Acquirer, DB Administrators, Software Maintainers and Developers.

Business Cases	Schedule-Driven	Market Trend-Driven	Product-Driven
Primary Objective	Rapid value by adding small extra functionalities	Rapid Market Share Occupation	Version upgrade with Q-attribute achievement: reliability, availability, performance, evolvability, etc.
Quality Risks	Low	Medium	High; major business losses
Schedule Risks	High; major business losses	High; market share loss	Low
Stakeholders	Single collocated representatives	Many success-critical stakeholders	Multiple success-critical stakeholders with various Q-attribute requirements
Requirements	1) A few specific and stable requirements 2) Mostly functional	1) Goals generally known (e.g., platform changes) 2) Detailed requirements often vague, volatile and emergent 3) Functional and non-functional [Chung 1999]	1) Critical and conflicting Q-attribute requirements from various stakeholders 2) Most requirements relatively stable; others volatile, emergent 3) Functional and nonfunctional[Chung 1999]
Architecture	1) Extend from existing system architecture 2) Little architecting effort 3) Stakeholder high confidence	1) Brand new architecture 2) Most architecting effort; 3) Stakeholder low confidence	1) Evolve based on existing product-line architecture 2) High confidence in some parts; low confidence in others
Refactoring	Inexpensive with skilled people	More expensive with mix of people skills	Very expensive, with mix of people skills

Table 5.16. Characteristics of Three Example Business Cases in ERP Software Development

5.3.6.2 Tailor VBSQA Process to Schedule-Driven Business Case

Schedule-driven business case applies when rapidly accommodating a few minor product upgrading requirements from one or two departments within an organization. The examples of such requirements can be adding, deleting, updating certain attributes in the current DIMS database schema. Those functionalities are usually needed urgently so that delivering the functionalities on time becomes the stakeholders' highest-priority value proposition. Thus, we need to prioritize the process steps/activities and tailor the VBSQA process framework to only retain the most effective process steps/milestones/activities. In this case, system users are willing to tolerate some quality degradation and delay the Q-attribute requirements until the system operation.

Based on the schedule-driven business case analysis in Table 5.16, the added functionalities are extended from the existing system architecture and the stakeholders are more confident in the architecture. Thus there is no need to propose or review several feasible architectural options. And requirements are specific enough to skip the high-level design and to proceed directly to the detailed design stage. In this case, the process steps in Life Cycle Objective (LCO) stage are less effective than those in the Life Cycle Architecture (LCA) stage in VBSQA process. For the same reason, we may also skip the intermediate milestone Core Capability Demo (CCD) and proceed to Initial Operational Capability (IOC) Readiness Review. Since the quality risks are relatively low and developers only need to extend from the existing system architecture, in LCA stage the *Selected Architectural Internal Review* within the developer team is per-formed instead of the *onsite External Review* with the participation of all success-critical stakeholders, which would have require more time and effort. Figure 5.12 shows an example of schedule-driven process strategy for DIMS project.

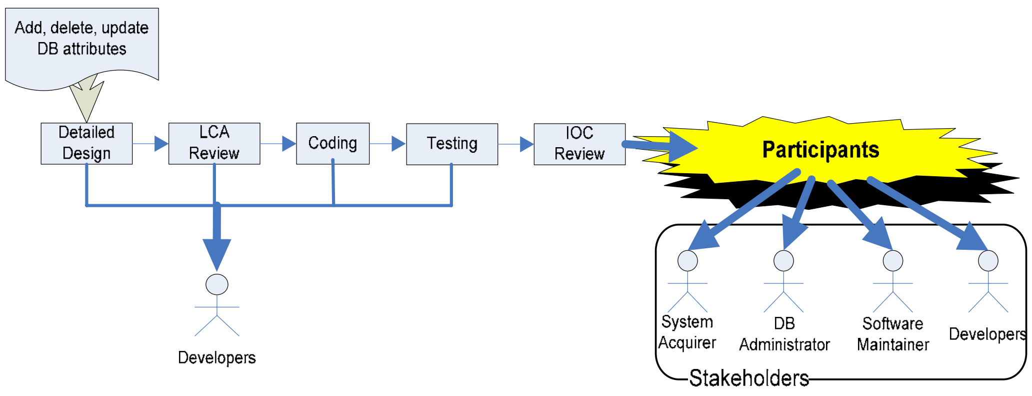


Figure 5.12. An Example of Schedule-driven Process Strategy for DIMS Project

5.3.6.3 Tailor VBSQA Process to Product-Driven Business Case

Product-driven business case applies when accommodating a request to upgrade to the next version due to the aggregation of some common upgrading requirements from various departments. In this case, the quality of the upgraded product is the process driver rather than meeting a schedule. Quality risks are dominant compared with schedule risks shown in Table 5.16. The requirements are relatively stable. Since the requirements are aggregated from various project stakeholders, the Q-attribute requirements may conflict with one another. An example is the DIMS version upgrade from 6.0 to 7.0. Functional requirements and their associated Q-attribute requirements were prioritized through stakeholder WinWin negotiation in Table 5.17.

Requirements	Description	Category	Priority
R1	Data migration from old DB platform to upgraded DB platform	Functional	High
R2	Data migration shall be completed within 1 day and within the storage space	Quality (Performance)	High
R3	Accommodate different DB platforms and schema in data migration	Quality (Evolvability)	Medium
R4	Add a printing function in DIMS system	Functional	High
R5	Build a unified log in user interface for different DIMS subsystems	Functional	Medium
R6	Improve search response time from 2 seconds to 0.5 seconds	Quality (Performance)	Medium

Table 5.17. Prioritized Requirements in DIMS Upgrade from 6.0 to 7.0

Multiple project increments can be proposed based on the priorities of requirements. A process instance is generated for each increment. Figure 5.13 shows an example of product-driven process strategy in DIMS upgrade project. R1, R2, R3 and R4 are grouped into the first increments due to their high priorities. R5 and R6 are grouped into the second increment. In product-driven business case, process instances of multiple increments can proceed concurrently since the functional and Q-attribute requirements are relatively stable. In each increment, process strategy shall place emphasis on involving stakeholders in identifying and resolving conflicting Q-attributes, concurrently identifying and mitigating Q-risks with architecture/technology evaluation. Thus, its iteration cycle is longer than schedule-driven process in order to address the quality risks and maintain the product-line architecture.

LCO/LCA reviews and CCD are all necessary to identify and mitigate Q-risks in each increment. It is also important to involve all success-critical stakeholders in the prototype evaluation and each milestone review (i.e., LCO, LCA, CCD, IOC). Therefore, performing the onsite *External Prototype Evaluation*, *Architecture Options External Review* and *Selected Architecture External Review* with the participation of the System Acquirer, DB Administrators, Software Maintainers and Developers, is more effective than their internal counterparts within developer team.

5.3.6.4 Tailor VBSQA Process to Market Trend-Driven Business Case

Market trend-driven business case applies when the upgrade of the product is driven by the market trend or competing companies' products, such as a change from Client/Server architecture to web-based architecture in the DIMS system. In this case, providing superior capabilities to capture greater market share as early as possible is the key process driver.

The priorities of schedule risks and quality risks are comparable for market trend-driven business case as shown in Table 5.16. Therefore, the process strategy for market trend-driven business case is a mixture of the schedule-driven and product-driven process strategies. It is similar to schedule-driven process strategy in that it maintains the short iteration cycle in the first project increment since meeting the product delivery deadline is critical for capturing the market share early. However, since stakeholders are less confident in the web-based architecture, it is different from schedule-driven process strategy in that stakeholders should be closely involved in the prototype evaluation and each milestone review (i.e., LCO, LCA, CCD, IOC) as shown in Figure 5.14. Furthermore, there can be multiple project increments.

It is similar to product-driven process strategy in that it emphasizes stakeholder involvement and multiple project increments can be proposed based on the stakeholders' priorities of functional and Q-attribute requirements. However, it is different from product-driven process strategy in that only the top-priority capabilities can be accommodated in the first increment (see Figure 5.14) based on the Schedule/Cost/Quality as Independent Variable (SCQAIV) process strategy [Boehm et. al. 2002]. Stakeholders are usually willing to tolerate some quality (e.g., performance, evolvability) degradation at the initial trial of the system. In addition, the process strategy of the following increments heavily depends on the market feedback of the product delivered in the first increment. Thus, there is a gap between each increment to wait for the market feedback. As the operation of new platform becomes stable with sufficient market feedback, product-driven process strategy can be applied in the following increments.

5.3.7 Conclusions and Discussion

As we discovered in our application experiences of VBSQA process, solving a problem in theory and in practice were very different. In spite of the practical difficulties in applying a new process in software industry where traditional processes and methods dominated, the results showed that the application of value-based approaches was inherently better than the value-neutral ones that most ERP software projects employed in China.

In Microsoft Secrets [Cusumano-Selby 1995], the ability to synchronize and stabilize multiple internal development teams is identified as a Microsoft critical success factor. The VBSQA-OPN model provided a framework in which the activities, value propositions, and commitments of multiple success-critical stakeholders could be synchronized and stabilized for a wide variety of process drivers.

The tailored VBSQA process based on project business case can be used as an input to the System Net (*SN*) of the VBSQA-OPN process model.

The experience with the VBSQA Process Generator also told us process visualization and simulation tools significantly increased management visibility and controllability for the success of software project. In order to build such tools to visualize, verify and simulate the value-based processes involved by various stakeholders, the Object Petri Nets (OPN) provided a feasible solution to the value-based process modeling.

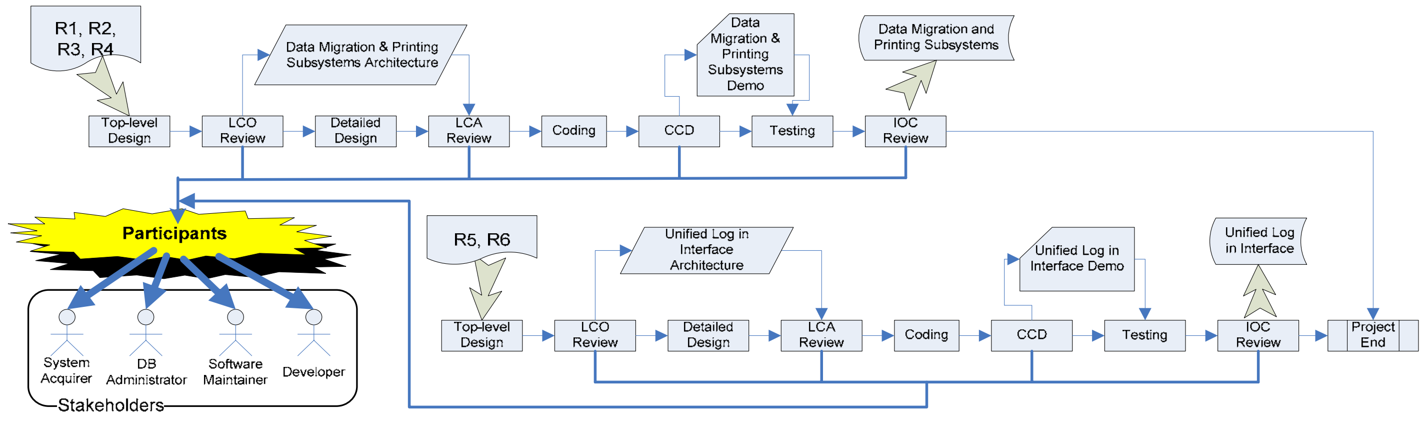


Figure 5.13. An Example of Product-driven Process Strategy for DIMS Version Upgrade

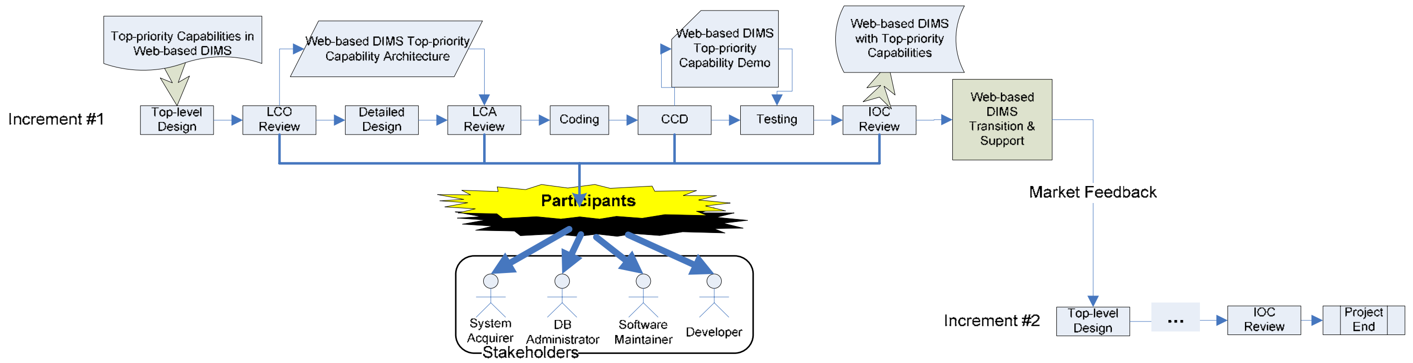


Figure 5.14. An Example of Market Trend-driven Process Strategy for Changing from Client/Server-based DIMS to Web-based DIMS

Chapter 6

Contributions and Future Research Challenges

6.1 Summary of Key Contributions

Despite the emergent of a large number of software quality improving techniques, in practice people tend to use the value-neutral approaches in software quality analysis and achievement. Based on the literature survey and empirical results, the traditional views of software quality often lead to stakeholder unsatisfactory outcomes even if sometimes an almost zero-defect product is delivered. The reason is that different systems have different success-critical stakeholders, and even for the same system these stakeholders may depend on it in different ways. There are no universal one-size-fits-all software quality metrics to optimize and we need to balance stakeholders' different value propositions on software quality attributes. Thus, there is an increasing need for stakeholder/value-based approaches to software quality modeling and its achievement monitoring and control.

The key contributions of this dissertation research can be summarized as follows: It proposes a Value-Based Software Quality Analysis framework, which consists of the definitions, metrics, model, and process to address various aspects of software and system quality analysis and achievement using value-based approaches. Specifically,

- The stakeholder/value-based definitions and metrics of software attributes differ from the traditional value-neutral ones in that they explicitly reflect the relevant success-critical stakeholders' value propositions and operational scenarios of a software system.
- Software quality investments compete for resources with investments in other project requirements and/or constraints. Weak quality business case analysis lead to inadequate or excessive investments. The VBSQM provide a technique for reasoning about the ROI of software quality attributes and performing combined risk analyses of both quality and market share erosion. It helps project decision-makers determine how much software quality investment is enough based on their project's business case. Furthermore, it provides a way to define appropriate quality attribute levels for different software classes or mission scenarios based on stakeholders' value propositions, which avoids the one-size-fits-all quality metrics for a software system.
- The combined risk analysis of VBSQM also shows that value-based software quality achievement techniques reduce the overall project risks.
- The Value-Based Software Quality Achievement (VBSQA) process, driven by the VBSQM and scenario-based approach, can be applied to determine whether a software system with stakeholder mutually satisfactory quality attribute requirements is achievable and to realize achievable stakeholder mutually satisfactory project outcomes. Furthermore, it enables us to

perform iterative value-based feedback control of the actual progress of a project’s quality achievement.

- In spite of the practical difficulties in applying a new process in software industry where traditional Waterfall processes and methods dominated, the application experience of VBSQA process in real-world ERP software development shows that the application of value-based approaches is inherently better than the value-neutral ones that most ERP software projects have employed in China.
- The Object Petri Nets (OPN) enables the separation of concerns among various stakeholders with different dependencies on quality attributes and different perspectives on software development process. In the meantime, it supports the synchronization and interaction among stakeholders when needed. Our successful experience of VBSQA-OPN modeling for ERP software development in China shows that OPN provides a feasible solution to the value-based process modeling.

6.2 Future Research Challenges

There are many useful extensions and future research that can be done to the VBSQM and the VBSQA process.

6.2.1 Future Research on the VBSQM

The suggestive extensions on the VBSQM can be:

- Develop and calibrate domain-oriented quality-attribute-estimating relationships (QERs) and value-estimating relationships (VERs) for additional quality attributes such as safety, security, performance, survivability, accuracy, etc.
- Calibrate and validate the VBSQM in various software project domains and in various organizations.

6.2.2 Future Research on the VBSQA Process

The suggestive future research on the VBSQA process can be:

- Apply the VBSQA process on more project domains.
- Investigate more tradeoff models among software quality attributes based on different project domains.
- The relationship between what a certain quality improving technology do (e.g., to remove certain classes of defects) and what stakeholders depend on (e.g. to achieve acceptable/desired levels of a quality attribute) is often not straightforward. Constructing a more specific bridge between them is worthy of further research effort.

- Create value-based counterparts for such value-neutral software quality technologies as test data generators, inspection checklists, defect closure metrics, and test plan aids.

6.2.3 Future Research on the VBSQA-OPN Model

- Support more simulations in VBSQA Process Simulator based on VBSQA-OPN model in order to provide more complete support for process decision-making.
- Compare OPN with other process modeling techniques (e.g., Little-Jil [Wise et. al. 2000]) in modeling value-based processes.

Bibliography

[Aalst 1998] W. M. P. van der Aalst, "The Application of Petri Nets to Workflow Management", Journal of Circuits, Systems, and Computers, 1998, pp. 21-66.

[Al-Said 2003] M. Al-Said, "Ph.D. Dissertation: Identifying, Analyzing, and Avoiding Software Model Clashes," USC-CSE, 2003.

[Arlat 2001] J. Arlat, "Dependability Benchmarking: The SIG Class/Factor/Criteria Framework," presentation at the 39th Meeting of IFIP WG 10.4, February 2001, Paraty, Brazil.

[Aurum et. al. 2005] A. Aurum, S. Biffl, B. Boehm, H. Erdogmus, and P. Gruenbacher (eds.), Value-Based Software Engineering, Springer Verlag, 2005.

[Austin 1996] R. D. Austin, Measuring and Managing Performance in Organization, Dorset House, 1996.

[Avizienis et. al. 2002] A. Avizienis, J.C. Laprie, B. Randell, "Fundamental Concepts of Computer System Dependability", in IARP/IEEE-RAS Workshop on Robot Dependability: Technological Challenge of Dependable Robots in Human Environments, May 2002, Seoul, Korea.

[Baldwin-Clark 2000] C. Baldwin, and K. Clark, Design Rules: The Power of Modularity, MIT Press, 2000.

[Basili 2004] V. Basili, P. Donzelli, and S. Asgari, "A Unified Model of Dependability: Capturing Dependability in Context", IEEE Software, November/December 2004, vol. 21, no. 6, pp. 19-25.

[Bass 1998] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, Addison Wesley, 1998.

[Boehm et. al. 1978] B. Boehm, J. Brown, H. Kaspar, M. Lipow, G. MacLeod, M. Merritt, "Characteristics of Software Quality", TRW Report to National Bureau of Standards, November 1973; TRW Software Series Report, TRW-SS-73-09. Also published by North Holland, 1978.

[Boehm 1989] B. W. Boehm, Software Risk Management, IEEE Computer Society Press, 1989.

[Boehm et. al., 1995] B. Boehm, P. Bose, E. Horowitz, and M. Lee, “Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach”, Proceedings of the 17th International Conference on Software Engineering (ICSE-17), IEEE Computer Society Press, Seattle, April 1995.

[Boehm 1999] B. Boehm, D. Port, "When Models Collide: Lessons from Software System Analysis," IT Professional, IEEE-CS, January/February 1999, pp. 49-56.

[Boehm 2000] B. Boehm, “Unifying Software Engineering and Systems Engineering”, Computer, vol.33, no. 3, March, 2000.

[Boehm et. al. 2000a] B. Boehm, C. Abts, A.W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Riefer, and B. Steece, Software Cost Estimation with COCOMO II, Prentice Hall, 2000.

[Boehm et. al. 2000b] B. Boehm, D. Port, M. Al-Said, “Avoiding the Software Model-Clash Spiderweb,” Computer, November 2000, pp. 120-122.

[Boehm 2001] B. Boehm, “Overview: USC Annual Research Review”, February, 2001.

[Boehm-Brown 2001] B. Boehm, and A.W. Brown, “Mastering Rapid Delivery and Change with the SAIV Process Model,” Proceedings, ESCOM 2001, April 2001.

[Boehm-Hansenzz, 2001] B. Boehm, W. Hansenzz, “Understanding the Spiral Model as a Tool for Evolutionary Acquisition”, CrossTalk, May, 2001.

[Boehm et. al. 2002] B. Boehm, D. Port, L. Huang, A. W. Brown, “Using the Spiral Model and MBASE to Generate New Acquisition Process Models: SAIV, CAIV, and SCQAIV”, CrossTalk, vol. 15, no. 1, January 2002.

[Boehm-Huang 2003] B. Boehm and L. Huang, “Value-Based Software Engineering: A Case Study”, IEEE Computer, vol. 36, no. 3, March 2003, pp. 33-41.

[Boehm-Huang 2004a] B. Boehm, L. Huang, A. Jain and R. Madachy, “The ROI of Software Dependability: The iDAVE Model”, IEEE Software, vol. 21, no. 3, May/June 2004, pp.54-61.

[Boehm-Huang 2004b] B. Boehm, L. Huang, A. Jain, and R. Madachy, “The Nature of Information System Dependability: A Stakeholder/Value Approach” (Draft 6)”, USC-CSE Technical Report, December, 2004.

[Boehm-Turner 2004] B. Boehm, R. Turner, Balancing Agility and Discipline, Addison Wesley, 2004.

[Boehm 2005] B. Boehm, "The Future of Software and Systems Engineering Processes", USC-CSE Technical Report 2005-507, June 2005.

[Bullock 2000] J. Bullock, "Calculating the Value of Testing," Software Testing and Quality Engineering, May/June 2000, pp. 56-62.

[Butler 2002] S. Butler, "Security Attribute Evaluation Method: A Cost-Benefit Approach", Proceedings ICSE 2002, pp. 232-240.

[Cai 2002] Y. Cai, K. Sullivan, "Stochastic Optimal Switching", Proc. 4th Workshop on Economics-Driven Software Engineering Research, IEEE CS Press, 2002.

[Chung 1999] L. Chung, B. Nixon, E. Yu, J. Mylopoulos, Non-Functional Requirements in Software Engineering, Kluwer, 1999.

[Clements 2002] P. Clements, R. Kazman, and M. Klein, Evaluating Software Architecture: Methods and Case Studies, Addison Wesley, 2002.

[Cockburn 2002] A. Cockburn, Agile Software Development, Addison Wesley, 2002.

[CODASYL 1976] CODASYL Systems Committee, "Selection and Acquisition of Data Base Management Systems", ACM, New York, March, 1976.

[Cusumano-Selby 1995] M. Cusumano and R. Selby, Microsoft Secrets, The Free Press, October, 1995.

[Deiters-Gruhn 1994] W. Deiters and V. Gruhn, "The FUNSOFT Net Approach to Software Process Management", International Journal on Software Engineering and Knowledge Engineering, 1994, pp. 229-256.

[Demillo 2001] R. DeMillo, "Why Software Falls Down", Mutation Testing for the New Century, W.E. Wong, ed., Kluwer Academic, 2001.

[Deming 1989] E. Deming, Out of Crisis, MIT Center for Advanced Engineering Study, Cambridge, MA, 1989.

[DoD 1974] U.S. Department of Defense, "Software Quality Assurance Program Requirements", MIL-S-52779 (AD), 5, April 1974.

[Emam 2003] Khaled El Emam, The ROI from Software Quality: An Executive Briefing, K Sharp Technology Inc., 2003.

[Eureka-Ryan 1988] W. E. Eureka and N. E. Ryan, The customer-Driven Company: Managerial Perspectives on QFD, Dearborn, Mich.: ASI Press, 1988.

[Feather 2002] M. S. Feather, S. L. Cornford, J. Dunphy, "A Risk-Centric Model for Value Maximization", Proceedings of 4th Workshop on Economics-Driven Software Eng. Research, IEEE CS Press, 2002.

[Flowers 1996] S. Flowers, Software Failure: Management Failure, Wiley, 1996.

[Gazebo 2005] "Gazebo: 3D Multiple Robot Simulator With Dynamics", http://playerstage.sourceforge.net/gazebo/gazebo.html.

[Gerrard 2002] P. Gerrard, N. Thompson, Risk-Based E-Business Testing, Artech House, Inc., 2002.

[Gilb 1969] T. SCHARF (GILB), "Weighted Ranking by Levels", IAG Journal, 2, 1969, pp. 7-23.

[Gilb 1976] T. Gilb, Software Metrics, Studentlitteratur AB, Lund, Sweden, 1976.

[Gilb 1988] T. Gilb, Principles of Software Engineering Management, Addison Wesley, 1988.

[Glass 1998] R. Glass, Software Runaways, Prentice Hall, 1998.

[Grady 1992a] R. Grady, D. Caswell, Software Metrics: Establishing A Company-wide Program, Prentice Hall, 1992.

[Grady 1992b] R. Grady, Practical Software Metrics for Project Management and Process Improvement, Prentice Hall, 1992.

[Highsmith 2002] J. Highsmith, Agile Software Development Ecosystems, Addison Wesley, 2002.

[Huang 2005] L. Huang, "A Value-Based Process for Achieving Software Dependability", Proceeding of 1st International Software Process Workshop, Beijing, May 2005.

[Huang-Boehm 2005a] L. Huang and B. Boehm, "Determining How Much Software Assurance Is Enough? A Value-based Approach", Proceeding of the 7th International Workshop on Economics-Driven Software Engineering Research (EDSER), May 2005.

[Huang-Boehm 2005b] LiGuo Huang and Barry Boehm, "Determining How Much Software Assurance Is Enough? A Value-Based Approach", Proceedings of 4th International Symposium on Empirical Software Engineering (ISESE), November, 2005.

[Huang-Boehm 2006] LiGuo Huang and Barry Boehm, “How Much Software Quality Investment Is Enough: A Value-Based Approach”, IEEE Software, vol. 23, no. 5, September/October, 2006, pp. 88-95.

[Huang et. al. 2006a] LiGuo Huang, Hao Hu, Jidong Ge, Barry Boehm, and Jian Lü, “Tailor the Value-Based Software Quality Achievement Process to Project Business Cases”, Proceedings of Software Process Workshop (SPW) 2006, May 2006, LNCS.

[Huang et. al. 2006b] LiGuo Huang, Barry Boehm, Hao Hu, Jidong Ge, Jian Lu and Cheng Qian, “Applying the Value/Petri Process to ERP Software Development in China”, Proceedings of 28th International Conference of Software Engineering (ICSE), May 2006.

[Huynh 2003] D. Huynh, M. Zelkowitz, V. Basili, and I. Rus, “Modeling Dependability for a Diverse Set of Stakeholders”, The International Conference on Dependable Systems and Networks, 2003 (DSN-2003).

[IEEE 1988a] IEEE Std 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software, 1988.

[IEEE 1988b] IEEE Std 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measuresto Produce Reliable Software, 1988.

[IEEE 1990] ANSI/IEEE Std. 610.12-1990, IEEE Standard. Glossary of Software Engineering Technology, IEEE Computer Society Press, 1990.

[IEEE 1992] IEEE Std 1061-1992, IEEE Standard for a Software Quality Metrics Methodology, 1992.

[IEEE 1994] IEEE Standard 1228-1994, Software Safety Plans, 1994.

[IEEE 1998] IEEE Standard 1219-1998, Software Maintenance, 1998.

[IFIP WG10.4] International Federation for Information Processing (IFIP WG-10.4), www.dependability.org.

[In 1998] H. In, “Conflict Identification and Resolution for Software Attribute Requirements”, USC Ph.D. Dissertation, 1998.

[In 2001] H. In, B. Boehm, T. Rodgers, M. Deutsch, “Applying WinWin to Quality Requirements: A Case Study”, Proceedings, ICSE 2001, ACM/IEEE, pp. 525-564.

[Kaplan-Norton 1996] Robert S. Kaplan and David P. Norton, The Balanced Scorecard: Measures That Drive Performance (HBR OnPoint Enhanced Edition), Harvard Business School Press, 2000.

[Keeney 1993] R. L. Keeney, H. Raiffa, Decisions With Multiple Objectives, Cambridge University Press, 1993.

[Kellner et. al. 1999] M. Kellner, R. Madachy, D. Raffo: “Software process simulation modeling: Why? What? How?”, Journal of Systems and Software, Vol. 46, No. 2/3, April, 1999.

[Kitchenham 1989] B. Kitchenham, L. Pickard, “Towards a Constructive Quality Model”, *Software Engineering Journal*, 1989, 2(4), pp. 114-126.

[Kleijnen 1980] J. Kleijnen, Computers and Profits: Quantifying Financial Benefits of Information, Addison Wesley, 1980.

[Koopman 1999] P. Koopman, H. Madeira, “Dependability Benchmarking and Prediction: A Grand Challenge Technology Problem”, in Real-Time Mission-Critical Systems: Grand Challenge Problems, Phoenix, Arizona, November 1999.

[Lapire 1992] Jean-Claude Lapire, “Dependability: Basic Concepts and Terminology, Dependable Computing and Fault Tolerance”, Vienna, Austria, Springer-Verlag, 1992.

[Laprie 1998] J. C. Laprie, “Dependability of Computer Systems: from Concepts to Limits”, IFIP International Workshop on Dependable Computing and its Applications, Johannesburg, January, 1998, pp. 108 –126.

[Lee 2005] Keun Lee, Barry Boehm, “Empirical Results from an Experiment on Value-Based Review (VBR) Processes”, 4th International Symposium on Empirical Software Engineering (ISESE), November, 2006.

[Leveson 1995] N. G. Leveson, Safeware, System Safety and Computers, Addison-Wesley Publishing Company, 1995.

[Li 2002] P. Li, M. Shaw, K. Stolarick, K. Wallnau, “The Potential for Synergy Between Certification and Insurance”, Special edition of ACM SIGSOFT from the ICSR7, April 2002.

[Lientz-Swanson 1978] B. Lientz, E. Swanson, and G. Tompkins, “Characteristics of application software maintenance”, Communications of the ACM, 21(6), June 1978.

[Linger 1996] Richard Linger and Carmen Trammell, “Cleanroom Software Engineering Reference Model Version 1.0,” Technical Report, CMU/SEI-96-TR-022, November 1996.

[Lyu 1996] Jean-Claude Laprie and Karama Kanoun, in Michael Lyu, ed., Handbook of Software Reliability Engineering, IEEE Computer Society Press, McGraw Hill, 1996, 27-69.

[Madachy-Lee 2003] Ray Madachy, Keun Lee, "Opportunity Trees Demo", HDCP Review, June 2003.

[Madeira 2001] H. Madeira and P. Koopman, "Dependability Benchmarking: Making Choices in an N-Dimensional Problem Space", the First Workshop on Evaluating and Architecting System dependability (EASY), July 2001, Goteborg, Sweden.

[Maslow 1954] A.H. Maslow, Motivation and Personality, New York, NY: Harper and Bros, 1954.

[MBASE 2003] Guidelines for Model-Based (System) Architecting and Software Engineering (MBASE), http://cse.usc.edu/research/MBASE**,** USC-CSE, 2003.

[McCall et. al., 1977] J. McCall, P. Richards, P. Walters, Factors in Software Quality, NTIS AD-A049-014, 015, 055, 1977.

[Nejmeh 2002] B. Nejmeh, I. Thomas, "Business-Driven Product Planning Using Feature Vectors and Increments", Software, November-December 2002, pp. 34-42.

[Osterweil 1987] L. J. Osterweil, "Software Processes are Software too", Proceedings of International Conference of Software Engineering, 1987, pp. 2-13.

[Padberg 2003] F. Padberg, "A Software Process Scheduling Simulator," Proceedings of 25th International Conference of Software Engineering (ICSE'03), May, 2003.

[Pardee 1996] W. Pardee, To Satisfy & Delight Your Customer: How to Manage for Customer Value, Dorset House Publishing, NY, 1996.

[Prowell 1999] Stacy Prowell, Carmen Trammell, Richard Linger, and Jesse Poore, Cleanroom Software Engineering: Technology and Process, SEI series in Software Engineering, Addison-Wesley, 1999.

[Radice 1985] R. Radice et. al., "A Programming Process Study", IBM Sys. J. 24(2), pp. 91-101, 1985.

[Ramler et. al. 2006] R. Ramler, S. Biffl, and P. Gruenbacher, "Value-Based Management of Software Testing," in A. Aurum, S. Biffl, B. Boehm, H. Erdogmus, and P. Gruenbacher, Value-Based Software Engineering, Springer Verleg, 2006.

[Raz 2001] O. Raz, M. Shaw, “Software Risk Management and Insurance”, Proceedings of the 23rd International Conference on Software Engineering (Workshop on Economics-Driven Software Engineering Research), 2001.

[Reifer 2002] D. Reifer, Making the Software Business Case, Addison Wesley, 2002.

[Reifer 2003] D. Reifer, B. Boehm, and M. Gangadharan, "Estimating the Cost of Security for COTS Software," Proceedings Second Intl. Conf. COTS-Based Software Systems, February 2003, pp.178-186.

[Reisig 1985] W. Reisig, Petri Nets, An Introduction, Springer Verlag, Berlin, 1985.

[Ross-Swanson 2003] R. Ross, M. Swanson, G. Stoneburner, S. Katzke, A. Johnson, Guide for the Security Certification and Accreditation of Federal Information Systems, NIST (National Institute of Standards and Technology) Special Publication 800-37, 2003.

[Rus et. al. 2002] I. Rus, V. Basili, B. Boehm, “ Empirical Evaluation of Techniques and Methods Used for Achieving and Assessing Software High Dependability”, Workshop on Dependability Benchmarking, in conjunction with the International Conference on Dependable Systems and Networks (DSN-2002), Washington, DC, June 2002.

[Rus et. al. 2003] I. Rus, S. Komi-Servio, P. Costa, “Software Dependability Properties: A Survey of Definitions, Measures and Techniques,” Fraunhofer Technical Report 03-110, January 2003.

[Saaty 1989] Saaty, T.L. and J. Alexander, Conflict Resolution: The Analytic Hierarchy Process, Praeger, New York.

[Schwaber and Beedle 2002] K. Schwaber and M. Beedle, Agile Software Development with Scrum, Prentice Hall, 2002.

[Shaw 2002] M. Shaw, “Everyday Dependability for Everyday needs”, Keynote on 13th International Symposium on Software Reliability Engineering (ISSRE), Nov. 2002.

[Snir 2003] M. Snir and D. A. Bader, “A Framework for Measuring Supercomputer Productivity", Technical Report, October 2003.

[Sommerville 2004] Ian Sommerville, Software Engineering (7th Edition), Chapter 20, Addison Wesley, May 2004.

[Standish 1995; Standish 2001] The Standish Group, CHAOS Report, 1995, 2001; www.standishgroup.com.

[Steece, et. al. 2002] B. Steese, S. Chulani, B. Boehm, “ Determining Software Quality Using COQUALMO,” Case Studies in Reliability and Maintenance, W. Blischke and D. Murthy, eds., Jon Wiley & Sons, 2002.

[Sullivan 1999] K. Sullivan, P. Chalasani, S. Jha, V. Sazawal, “Software Design as an Investment Activity: A Real Options Perspective”, in Real Options and Business Strategy: Applications to Decision Making, L. Trigeorgis, consulting editor, Risk Books, December 1999.

[Sullivan et. al. 1999-2005] K. Sullivan et. al., The International Workshop on Economics-Driven Software Engineering Research (EDSER), affiliated each year with the International Conference of Software Engineering, 1999-2005. See http://www.edser.org.

[Sullivan et. al. 2001] K. Sullivan, Y. Cai, B. Hallen, W. Griswold, “The structure and value of modularity in software design” 3rd International Workshop on Economics-driven Software Engineering Research (EDSER), May, 2001.

[Thorp 1998] J. Thorp and DMR, The Information Paradox, McGraw Hill, 1998.

[Tockey 2004] S. Tockey, Return on Software, Addison Wesley, 2004.

[Valk 1998] R. Valk, “Petri nets as token objects: An introduction to elementary object nets”, Proceedings of Application and Theory of Petri Nets, Springer-Verlag, 1998, pp. 1-25.

[Weinstock 2004] C. B. Weinstock, J. B. Goodenough, “Dependability Cases”, Technical Note, CMU/SEI-2004-TN-016, May 2004.

[West 2004] M. West, Real Process Improvement Using the CMMi, CRC Press, Feb. 1, 2004.

[Wilson 2002] D. Wilson, B. Murphy and L. Spainhower, “Progress on Defining Standardized Classes for Comparing the Dependability of Computer Systems”, Workshop on Dependability Benchmarking, in conjunction with the International Conference on Dependable Systems and Networks (DSN-2002), Washington , DC, June 2002.

[WinWin 2006] WinWin Spiral Model & Groupware Support System, http://sunset.usc.edu/research/WINWIN/index.html.

[Wise et. al. 2000] Alexander Wise, Aaron G. Cass, Barbara Staudt Lerner, Eric K. McCall, Leon J. Osterweil, Stanley M. Sutton, Jr., “Using Little-JIL to Coordinate Agents in Software Engineering”, Proceedings of the Automated Software Engineering Conference (ASE 2000), Grenoble, France, pp. 155-163, September 2000.

Appendices

Appendix A: Behavior Analysis of COCOMO II RELY Ratings

The correspondence between COCOMO II RELY ratings and COQUALMO defect removal profile ratings is based upon a mapping between the behavior analysis behind the COCOMO RELY effort multiplier and the rationales of COQUALMO defect removal profile ratings. Table A.1 indicates the differences in project activities which will result from having a higher or lower required reliability (RELY).

RELY Ratings	Requirements and Product Design	Detailed Design	Code and Unit Test	Integration and Test
Very Low	Little detail Many TBDs Little verification Minimal QA, CM, draft user manual, test plan Minimal PDR	Basic design information Minimal QA, CM, draft user manual, test plans Informal design inspections	No test procedures Minimal path test, standards check Minimal QA, CM Minimal I/O and off-nominal tests Minimal user manual	No test procedures Many requirements untested Minimal QA, CM Minimal stress, off-nominal tests Minimal as-built documentation
Low	Basic information, verification Frequent TBDs Basic QA, CM, standards, draft user manual, test plans	Moderate detail Basic QA, CM draft user manual, test plans	Minimal test procedures Partial path test, standards check Basic QA, CM, user manual Partial I/O and off-nominal tests	Minimal test procedures Frequent requirements untested Basic QA, CM, user manual Partial stress, off-nominal tests
Nominal	Nominal project V&V	→	→	→
High	Detailed verification, QA, CM, standards, PDR, documentation Detailed test plans, procedures	Detailed verification, QA, CM, standards, CDR, documentation Detailed test plans, procedures	Detailed test procedures, QA, CM, documentation Extensive off-nominal tests	Detailed test procedures, QA, CM, documentation Extensive stress, off-nominal tests
Very High	Detailed verification, QA, CM, standards, PDR, documentation IV&V interface Very detailed test plans, procedures	Detailed verification, QA, CM, standards, CDR, documentation Very thorough design inspections Very detailed test plans, procedures IV&V interface	Detailed test procedures, QA, CM, documentation Very thorough code inspections Very extensive off-nominal tests IV&V interface	Very detailed test procedures, QA, CM, documentation Very extensive stress, off-nominal tests IV&V interface

Table A.1. Product Activity Differences Due to Required Software Reliability (RELY)

Appendix B: COQUALMO Defect Removal Profiles

Rating	Automated Analysis	Peer Reviews	Execution Testing and Tools
Very Low	Simple compiler syntax checking.	No peer review.	No testing.
Low	Basic Compiler capabilities for static module-level code analysis, syntax, type-checking.	Ad-hoc informal walk-throughs Minimal preparation, no follow-up	Ad-hoc testing and debugging. Basic text-based debugger.
Nominal	Some compiler extensions for static module and inter-module level code analysis, syntax, type-checking. Basic requirements and design consistency, traceability checking.	Well-defined sequence of preparation, review, minimal follow-up. Informal review roles and procedures.	Basic unit test, integration test, system test process. Basic test data management, problem tracking support. Test criteria based on checklists.
High	Intermediate-level module and inter-module code syntax and semantic analysis. Simple requirements/design view consistency checking.	Formal review roles with all participants well-trained and procedures applied to all products using basic checklists, follow up.	Well-defined test sequence tailored to organization (acceptance/alpha/beta/flight/etc.) test. Basic test coverage tools, test support system. Basic test process management.
Very High	More elaborate requirements/design view consistency checking. Basic distributes-processing and temporal analysis, model checking, symbolic execution.	Formal review roles with all participants well-trained and procedures applied to all product artifacts & changes (formal change control boards). Basic review checklists, root cause analysis. Formal follow-up. Use of historical data on inspection rate, preparation rate, fault density.	More advanced test tools, test data preparation, basic test oracle support, distributed monitoring and analysis, assertion checking. Metrics-based test process management.
Extra High	Formalized* specification and verification. Advanced distributes processing and temporal analysis, model checking, symbolic execution. *Consistency-checkable pre-conditions and post-conditions, but not mathematical theorems.	Formal review roles and procedures for fixes, change control. Extensive review checklists, root cause analysis. Continuous review process improvement. User/Customer involvement, Statistical Process Control.	Highly advanced tools for test oracles, distributed monitoring and analysis, assertion checking Integration of automated analysis and test tools. Model-based test process management.

Table B.2. Rationales of COQUALMO defect removal profile ratings

Appendix C: Empirical Analysis on Stakeholder/Value Dependency on Quality Attributes in Information Systems

● Information System Quality-Attribute-Dependency-Differing Stakeholder Classes

The empirical analyses on USC-CSE e-service projects summarizes the stakeholder classes involved with an information system to have different patterns of dependency on quality attributes. The stakeholder classes are shown in Figure C.1 and the role of each stakeholder class is described as follows.

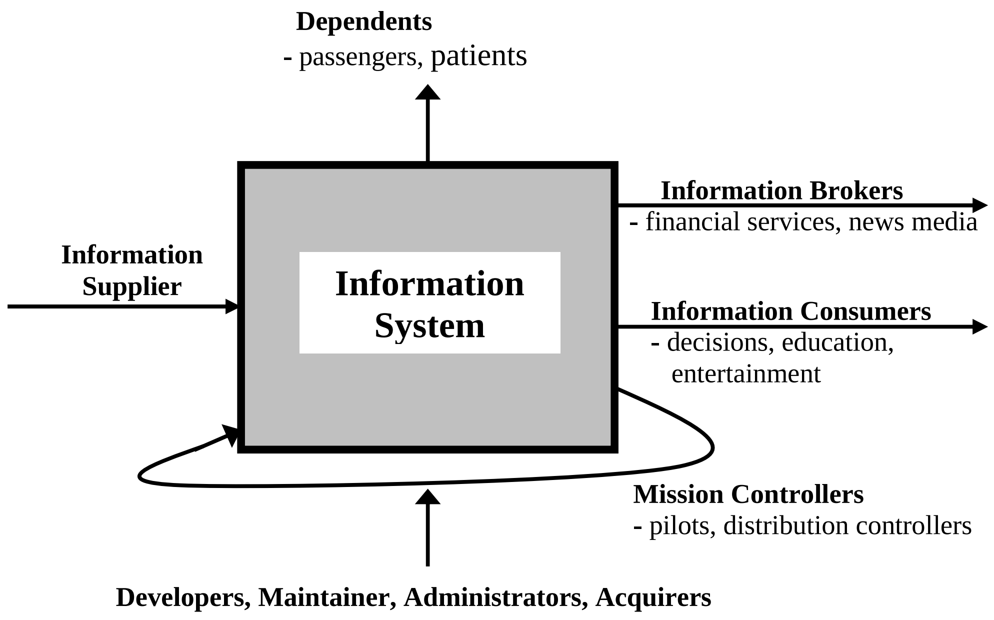

Dependents
- passengers, patients
Information Brokers
- financial services, news media
Information Supplier
Information System
Information Consumers
- decisions, education, entertainment
Mission Controllers
- pilots, distribution controllers
Developers, Maintainer, Administrators, Acquirers

Figure C.1. Major Information System Stakeholder Classes

Information Suppliers. These stakeholders have supplied information that will be used by other classes of stakeholders. They may be either individuals (medical patients, Internet shoppers, others) or organizations (companies filing tax data, submitting contract proposals, others). In their role as information suppliers, they will depend directly on the information system to protect their information's privacy and security. They may depend on other attributes when they also play other roles (such as information consumers when acting as Internet shoppers), but as information suppliers, their only other critical dependency is on the accuracy of the data about themselves in the information system. And, not all information suppliers are concerned with privacy, e.g., advertisers, other public information sources.

Information Consumers. These stakeholders consume information supplied by an information supplier or a broker. In conjunction with the above given examples, this stakeholder class includes individuals (doctors, sales executives, movie-goers, others) or organizations (the revenue service, retail stores, others). The stakeholders in this class depend on quite a few attributes such as the timeliness, trustworthy provenance and accuracy of the information given to them by an information supplier. We have discussed earlier the challenges in generalizing a stakeholder class with respect to a set of attributes that are of concern to them; as operational contexts vary, so will these attributes. Although this is true for all stakeholder classes, we have further categorized this stakeholder class into two extreme contexts: mission-critical information consumers (e.g., doctors) vs. mission-uncritical information consumers (e.g., moviegoers).

Information Brokers. The stakeholders in this class consume information that is often reanalyzed to produce a more refined and customized set of information to their respective consumers. There are individuals (stockbrokers, journalists, consultants, others) and organizations (brokerage companies, search engine companies, consulting companies, others) that normally belong to this class. We also note that, albeit the stakeholders in this class fall into the two aforementioned classes as consumers of information and suppliers of information, we chose to specialize them as brokers. This is because our analysis in Table C.3 found that brokers' value dependencies were not just a simple combination of information-supplier and information-consumer dependencies.

System Dependents. Stakeholders in this class are people that are not involved in the system's development or operation, but are dependent on some of its attributes. These people could be either airplane passengers or medical patients and they are normally concerned about the system's safety, security and survivability. As an example of time-varying priorities with improved flight safety engineering and increased terrorist threat levels, we have begun to see an increased emphasis on security as well as on survivability and safety in the concerns of these system dependents.

System Controllers. These stakeholders such as airplane pilots, spacecraft mission controllers, or electric power distribution controllers, perform real-time control of a system. Value dependencies of such system controllers are analogous to those of their system dependents and end users. Thus, safety is a critical concern for airplane pilots and manned spacecraft mission controllers, but not for unmanned spacecraft mission controllers. Most real-time mission control systems do not have privacy concerns.

Administrators. These stakeholders are responsible for monitoring and managing system resources, security operations, user requests, system backup, etc. Typically administrators are also a type of system user. Their direct concerns with respect to reliability, accuracy and availability for doing their own job are significant but not critical. Their indirect concerns with these and safety, security, and survivability are critical only when they are critical for their users and system dependents. Administrators are also concerned about interoperability to the extent that the successful operation of the administrator's system depends on successful data interchange and control coordination with external systems.

Developers. These stakeholders are individuals or organizations that perform development activities such as requirements analysis, design, programming, and testing during the system's life cycle process. Developers' direct value dependencies on the system are primarily for sufficient reliability, availability, performance, etc. to perform their development functions correctly within cost and schedule constraints. Their concerns with safety, security, etc. derive from the value dependencies of their end users. Their concerns with reusability reflect its ability to facilitate their future development capabilities.

Maintainers. These stakeholders are responsible for making changes to the system for corrective, adaptive, perfective and preventive maintenance. That is, to make the system work right and often work better. Dependability of a system for a maintainer's direct functions imply a system that has rare and easily recoverable failures; performs efficiently; is easy to modify and evolve; and for which reusability can be leveraged, as with developers.

Acquirers. These stakeholders are procurers of a system, software product, or software service from a supplier as a customer. Their direct value dependencies are that the delivered capabilities correctly implement end user requirements within cost and schedule constraints, and are also easy to evolve and interoperate with co-dependent systems, and perhaps have product-line reusable components. Their concerns with Protection, Robustness, and Quality of Service attributes will derive from those of their system end users and dependents.

Note that a stakeholder may belong to multiple classes. For instance, an airline pilot is both a system controller and a system-dependent passenger.

- **Information System Top-Level Stakeholder/Value Dependency on Quality Attributes**

Table C.3 provides a top-level summary of the relative strengths of dependency on information system quality attributes, for classes of information system stakeholders exhibiting different patterns of strengths of dependency. Its initial portions were obtained from empirical analysis of different classes of information system stakeholders' primary concerns during win-win requirements negotiations. The dependency ratings refer only to direct dependencies. For example, system developers, acquirers, and administrators are concerned with safety or security only to the extent that a system's information suppliers, users, and dependents are concerned with them. And information suppliers and system dependents are only concerned with reliability and availability to the extent that these help provide their direct concerns with security and safety.

Quality Attributes	Stakeholder Classes									
	Info. Supplier	System Dependents	Info. Brokers	Info. Consumers		System Controllers	Developers	Maintainers	Administrators	Acquirers
				Mission -						
Protection				Critical	Uncritical					
Safety		**		**		**				
Security	*	**	**	**		**				
Privacy	**		**	*	*					
Robustness										
Reliability		*	*	**		**		*	*	
Availability		*	*	**		**		*	*	
Survivability		*	*	**		**		*	*	
Quality of Service										
Performance			**	**	*	**		*	*	
Accuracy, Consistency	**		**	**	*	**			*	
Usability	*		*	**	**	**			*	
Evolvability			*	**	*	*		**	*	**
Interoperability			**						*	**
Correctness							*			**
Cost							*			**
Schedule			*	**	*	*	**			**
Reusability							**	*		*

**** Critical   * Significant   ( ) Insignificant or indirect**

Table C.3. Information System Top-Level Stakeholder/Value Dependencies on Quality (Q-) Attributes

The starting point for the stakeholder/value dependency ratings in Table C.3 was an empirical analysis of stakeholder win-win negotiations between student development teams and USC campus service providers negotiating the requirements for a Web-based capability to improve their campus services [In 2001] shown as Figure 2.2 in Chapter 2. The thin lines and dotted lines in Figure 2.2 reflect the study's initial hypotheses about critical stakeholder/value dependencies. The dotted lines show hypothesized dependencies that were not borne out by the data. These have some claim to be special cases: the developers were student teams working for free and not concerned with labor costs, although the fixed 2-semester schedule was a critical concern. And since they were temporary workers developing one-of-a-kind systems, they were not much concerned about producing reusable software. The broad arrows were additional Customer dependencies on Assurance (Protection and Robustness), Interoperability, and Usability reflecting their concerns as administrators of the delivered system.

As in Table C.3, Figure 2.2 only records the direct concerns of the stakeholders with quality attributes. For example, the developers of the digital library systems did not furnish win conditions about Assurance, Interoperability, Usability, and Performance. They were generally furnished by the Customers and Users with direct value dependencies, and thereby became Developer concerns for the desired system.

Table C.4 provides a mapping between the stakeholder classes in Table C.3 and stakeholder categories in Figure 2.2 The interoperability category is more extensive than just the set of Information Brokers associated with an information system. It can also include other external co-dependent systems furnishing or being furnished with needed data and services. For example, interoperating systems associated with the digital library applications included the commercial SIRSI system used for main campus library services, three interlibrary loan systems, and several national archives operated by such agencies as the National Library of Medicine, the Securities and Exchange Commission, and the Dissertation Abstracts service. The customers involved in the digital library applications were campus librarians or archive administrators acting as both system Acquirers and Administrators.

Stakeholder Classes in Figure 2.2	Stakeholder Classes in Table C.3
General Public	Information Suppliers, System Dependents
Interoperator	Information Brokers
User	Information Brokers, Consumers and System Controllers
Developer	Developers
Maintainer	Maintainers
Customer	Administrators, Acquirers

Table C.4. Mapping of Stakeholder Classes in Table C.3 and Figure 2.2

Table C.5 provides the counterpart mapping from the top-level quality attributes in Figure 2.2 to the quality attributes in Table C.3. The Assurance attribute in Figure 2.2 covered the Protection, Robustness, and Correctness attributes in Table C.3, while the other categories covered other quality attributes for which some classes of stakeholders exhibited significant value dependencies.

Quality Attributes in Figure 2.2	Quality Attributes in Table C.3
Assurance (Dependability)	Safety, Security, Privacy, Reliability, Availability, Survivability, Accuracy, Correctness
Interoperability	Interoperability
Usability	Usability
Performance	Performance
Evolvability and Portability	Evolvability
Cost, Schedule	Cost, Schedule
Reusability	Reusability

Table C.5. Mapping of Quality Attributes in Figure 2.2 into Quality Attributes in Table C.3

● Top-level Stakeholder/Value Dependency Patterns in Table C.3

1. Some of the attributes have the same rating patterns for all the stakeholder classes, but are still individually important.

An example involves the Robustness attributes: reliability, availability, and survivability. One can have a system with a very high reliability or Mean Time Between Failures (MTBF), but if a database crash requires a week to repair, the loss of availability can be severe. The survivability metric is particularly important in hard-to-fix situations. As one example, in many distributed-processing applications, it is difficult to reproduce the cause of failure, increasing the value of a self-recovery or degraded-operation capability. As other example, unattended operations such as interplanetary spacecraft have limited recovery options, increasing the value of self-recovery capabilities.

2. There are no pairs of stakeholder classes that have the same rating patterns for all of the attributes.

This is the case by design: separate stakeholder classes with the same rating patterns for all of the attributes were combined, for example, there are several classes of administrators: database administrators, system administrators, application administrators. When these turned out to have the same overall rating patterns, they were combined for simplicity. On the other hand, roles merge in many cases. In some organizations, developers are also the maintainers – and for evolutionary development, the distribution is blurred. In some organizations such as startup companies, developers are also the acquirers. In such cases, the combined roles generally assume the most critical rating for each attribute.

3. Stakeholders with indirect value dependencies generally assume the criticality ratings of the stakeholders they are depending on.

Although one can't tell directly from the role of an acquirer, developer, or maintainer their strength of dependency on making a product safe or secure, one can usually assume that it reflects the strength of dependency of their operational stakeholders on safety or security. On the other hand, although a COTS user may not care what the cost of developing the COTS product may have been, they will usually care about the product price, which may or may not be well-correlated with its development cost. And operational stakeholders relying on maintainers with limited resources to satisfy their need for new capabilities will be concerned about the relative cost of the capabilities. Thus stakeholder/value dependencies will vary by situation or life cycle stage, confirming that the criticality levels are more suggestive than universally definitive.

- **Capabilities Provided by the Stakeholder/Value Dependency Analysis**

Even with the complexities and challenges during its application, the stakeholder/value dependency analysis provides some much-needed capabilities for reasoning about software quality attributes by capturing the stakeholders' value propositions in project scoping. These capabilities include:

- ➢ The framework corroborates the statement above that the universal one-size-fits-all quality metrics are unachievable in most project situations.

- It highlights the importance of using operationally representative stakeholders and scenarios in prioritizing and evaluating a software system’s quality attributes.
- It provides first-order guidance on which stakeholder classes to consult in determining a system’s quality-attribute priorities.
- It explicitly identifies sources of complexity in software quality assessment, and helps avoid the measurement dysfunction accompanying overly simplistic quality improvement initiatives.
- It provides the basis for developing specific processes for stakeholder-oriented quality achievement.