A VALUE-BASED THEORY OF SOFTWARE ENGINEERING

by

Apurva Jain

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)

May 2008

Acknowledgements

How far that little candle throws his beams! Make that three candles, follow the beam, and you have a dissertation. I have been most fortunate to have such three distinct guides that have carried me through in times of need, and lead me towards success. Professors Barry Boehm, Stan Rifkin and Paul Adler, without you I would not have been sitting today, writing this final page in concluding a degree that only few can dream, and fewer actually achieve.

Professor Barry Boehm, the ways and things an individual can learn from you is no less than the number of times "risk" has been used in the literature of software engineering. This dissertation is only a part of all I have learned from you, and I dedicate it to you with a commitment to further this research in the coming years.

Professor Stan Rifkin, the intersection at which engineering and social sciences meet, I thought was a place not known to any until you helped me find it. The journey was very rough, but I would not have seen the end of this dissertation if not for your kindness, compassion, and wonderful insights that kept me going.

Professor Paul Adler, I have had the privilege to know the likes of Marx and Weber through your eyes. From you I have learnt to uncover knowledge at its finest granularity, without which any research including mine would lack its depth.

My Family, you are my family, what more can I say. I had once set a goal to build a strong foundation for myself that is rich with education and research. This dissertation is some evidence in fulfilling that goal. And all along, it was no luck, but your love and support through and through.

Friends, you have brought unparalleled happiness and excitement in almost all the things I have done in life, including this dissertation. Alex, Jesal, Nikunj, Shamik, Steve – I thank you for your friendship and for the various forms in which you have chimed with me in concluding this dissertation.

Thank you all.

Table of Contents

List of Tables

List of Figures

Abstract

The activity of developing software systems is not an end, but a means to an end for the people who directly or indirectly depend on them. Taking such a view thus implies that software systems must be engineered such that they help people meet their ends. The primary focus of this study is to show how decisions about software systems can be made such that they are better aligned to realize the values (ends) of its stakeholders (people).

This study develops an interdisciplinary theory and process by integrating research in organization design, economics, and software engineering. It goes beyond both that in addressing why a software system is being produced, and how well it needs to perform. It makes an inquiry into the current practices related to decision making in thinking about and constructing or acquiring software systems. With a few exceptions, it shows how most current practices fall short in addressing the full set of stakeholder values (disconnected from stakeholder values), and how current practices use an insufficient unit of analysis that does not include the context in which the software has to transition (disconnected from context).

Chapter 1: Introduction

What is the primary thesis of this research?

The activity of developing software systems is not an end, but a means to an end for the people who directly or indirectly depend on them. Taking such a view thus implies that software systems must be engineered such that they help people meet their ends. The primary focus of this study is to show how decisions about software systems can be made such that they are better aligned to realize the values (ends) of its stakeholders (people).

Note that this is consistent with Webster's definition of "engineering":

(1) the activities or function of an engineer

(2a) the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people

(2b) the design and manufacture of complex products.

The main definition (2a) puts making things useful to people (i.e., satisfying their value propositions) at the center of what an engineering discipline should do. And the last definition is the one that applies to software engineering.

Research on software systems has been roughly split into two aspects – process and product. Product research is mainly focused on the "what is produced" – generally technology focused, and addresses problems mostly within

the ambit of computer science. These may include inventing technologies to improve a software system's quality attributes, such as performance and security, or identifying better approaches to implementing existing or new technologies. For example, Web Services was invented in response to the growing interoperability needs among software systems.

Process research instead addresses "how the product is being produced" – in particular, the managerial and administrative aspects of developing software systems. The focus is on the practices that guide the development of a software system from its conceptualization through its evolution to a possibly much larger system such as the kind that serves an enterprise. Examples of research within the process area may include methods for decision making, life cycle choice, cost and schedule estimation, design and analysis, and risk identification and management.

This study integrates product and process research. It goes beyond both in addressing "why" the product is being produced, and "how well" it needs to perform. It makes an inquiry into the current practices related to decision making in thinking about and constructing or acquiring software systems. With a few exceptions, it shows how most current practices fall short in addressing the full set of stakeholder values (practices disconnected from stakeholder values), and how current practices use an insufficient unit of analysis that does not include the context in which the software has to transition (practices disconnected from context). Some of these current practices are discussed next to set the context for

the primary thesis of this research: that value-based and context-connected approaches can produce more stakeholder-satisfactory outcomes. Others will be described in Chapter 2 – Literature review.

In addressing these issues, this study describes a proof of concept theory for developing software systems that uses stakeholder values as its unit of analysis. A significant contribution of this study is the explanation of the theoretical constructs used in creating a new theory, and a process framework that helped in making an initial assessment of its sufficiency and fitness with respect to a set of criteria.

Since this research deals with imprecise and situation-varying human values, a combination of analytic and qualitative approaches was chosen for assessment of its findings. Future work to strengthen this research should include quantitative assessments and further extensions for each of the key theoretical constructs.

Software Systems – State of Affairs

What is the motivation for this study?

The low rate of successful software systems (Standish Group, 2004); its outcomes such as loss of life (Finkelstein and Dowell, 1996; Leveson and Turner, 1993); loss of revenue (NIST, 2002; Flowers, 1996; Babcock, 1985; Carr 2002) corporate embarrassment (LA Times, 1987; WS Journal, 1989) has been both a reflection of

the field's lack of understanding of how best to develop software systems and an inspiration for this research to uncover better ways.

Table 1 (ranked from highest to lowest) is a compilation of the top-*n* risks from Boehm ("Boehm's Top 10 Risk List", accessed March, 29, 2007) and Jones (1994) that have been known to impact the success of a software system. These risks

Table 1. Top Software System Risks

Boehm (2002)	Boehm (2007)	Jones (1994)
Schedules, budgets, process	Architecture complexity; quality tradeoffs	**MIS**
Requirements Changes	Requirements volatility; rapid change	Creeping user requirements
Personnel Shortfalls	Acquisition, contracting process mismatches	Excessive schedule pressure
Requirements Mismatch	Customer-developer-user team cohesion	Low quality
Rapid change	Budget and schedule constraints	Cost overruns
Architecture, ilities	Requirements mismatch	Inadequate configuration control
Commercial off-the-shelf, external components	Personnel shortfalls	**Commercial**
Legacy Software	COTS and other independently evolving systems	Inadequate user documentation
Externally-performed tasks	Migration complexity	Low user satisfaction
User interface mismatch	User interface mismatch	Excessive time to market
		Harmful competitive actions
		Litigation expense

inform us that today the risks related to deficiencies in process guidance are far greater than in the underlying technology of a software system. Since this study is in the realm of process research, it hopes that it can at least positively impact some of these risks.

Overview of the Problem

What are the shortcomings of current practices?

There are many possible reasons and explanations for such a dismal state of affairs in software systems development. This study offers two explanations of some of these problems. First, current practices are largely disconnected from the context (organization and its environment) – that is, their scope (unit of analysis) is overly limited. Second, current practices are disconnected from stakeholder values – that is, there is little process guidance on how best to address stakeholder values in making tradeoffs when the desired properties (feature sets, security, ease of use) of a system are either conflicting, competing or dependent on each other for limited resources.

Context Disconnect

How does a disconnected context impact software systems?

Many software projects fail by succumbing to the "Field of Dreams" syndrome (Boehm and Turner, 2007). This refers to the American movie in which a Midwestern farmer hears a voice saying that if he builds a baseball field on his

farm, the legendary players of the past will appear and play on it ("Build the field and the players will come"). This syndrome – build the system and user will come -- now runs deep in much of the techniques and methods that are used in the process of developing software systems: they do not address the organizational context.

Developing a software system based on a set of formal specification does not guarantee that the system will generate the expected stakeholder values. The explicit connection between the "software" and the organizational "context" must be made. Thorp (1998) and his employer, the DMR Consulting Group, have developed a Benefits Realization Approach (BRA) to show how software initiatives must be connected with the organizational context to be successful in realizing the expected benefits. This study extends their approach by requiring that the success-critical stakeholders be explicitly identified in the context.

Figure 1 shows the organizational context of an order-processing system that a company had set out to develop in response to the shortcomings of its existing systems. The figure shows how a benefits chain connects a software system to its context by linking software system Initiatives (e.g., implement a new order entry system for sales) to Contributions (not delivered systems, but their effects on existing operations) and Outcomes, which may lead either to further contributions or to added value (e.g., increased sales). It also helps in uncovering assumptions usually hidden in the organizational context on which the software

system's success depends. For example, if the market stagnates due to an unforeseen event, the reduced time to deliver the product will not result in increased sales.

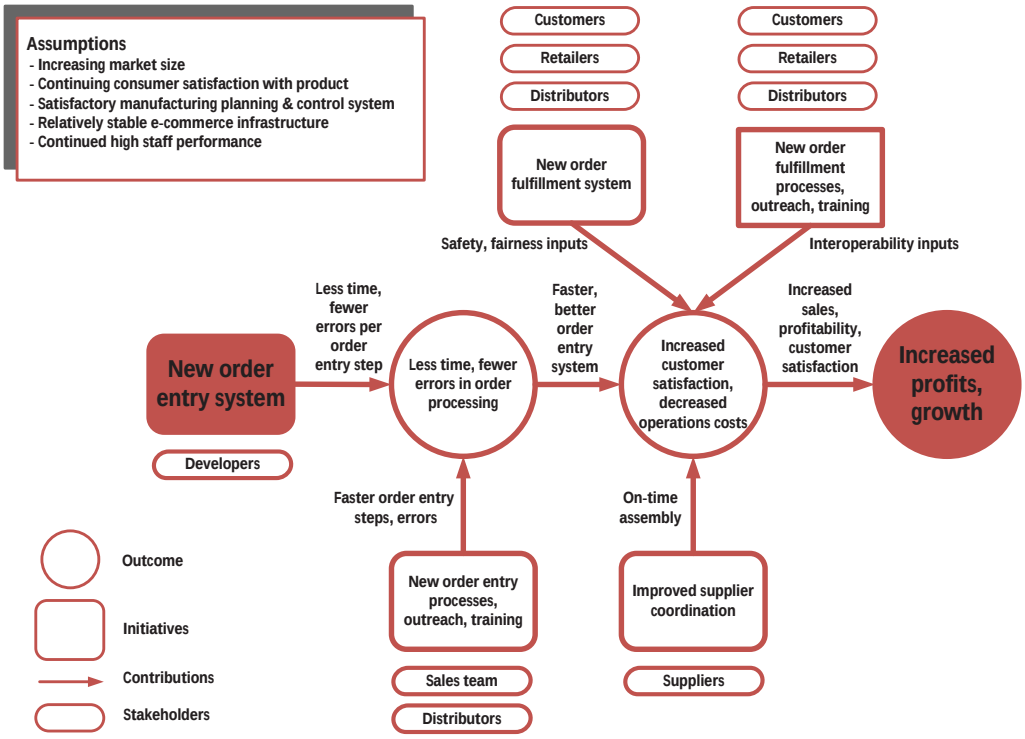


Figure 1. Benefits Chain as a Software System's Context
Adapted from (Boehm and Jain, 2007)

By including the organization context, a software system's project members can work with their stakeholders to identify additional non-software initiatives that may be needed to realize the stakeholder values that are associated with the software system initiative. Context also helps in identifying some additional success-critical stakeholders who need to be represented and "bought into" the project team.

For example, the initiative to implement a new order entry system may reduce the time required to process orders only if an additional initiative to convince the sales people that the new system will be good for their careers and to train them in how to use the system effectively is pursued. If the order entry system is so efficiency-optimized that it doesn't keep track of sales credits, the sales people will fight using it, so increased sales may also required adding a feature to keep track of sales credits so sales people will want to use the new system. Further, the reduced order processing cycle will reduce the time to deliver products only if additional initiatives are pursued to coordinate the order entry system with the order fulfillment system.

The benefits chain is one of many ways to connect a software system with its context. Another approach using contingency theory from Burton and Obel (2004) is described in the next section and in Chapter 3.

Stakeholder Value Disconnect

How are stakeholder values disconnected from practices?

Decisions about software systems are usually made in dimensions such as cost, schedule, quality, features, good practice vs. bad practice, technically sound, or methodology driven. Unfortunately, these by themselves have nothing to say about stakeholder values. "You can't control what you can't measure" (DeMarco, 1986, quoting Lord Kelvin) and "You only get what you measure" (Austin, 1996)

are the two of the most repeated slogans of measurement gurus. Still yet, current practices have remained disconnected from stakeholder values.

For example, acceptance testing is usually the last step in a system's development life cycle. It is performed by a customer prior to accepting delivery or accepting transfer of ownership. It involves a set of software-oriented tests approved by the customer as qualification criteria for successful development. Thus an acceptance test also serves as a contract between the developers and the customer.

From a legal sense, this is certainly sound. However, in a software system, such as an order processing system that was developed to reduce operational costs, increase market share, increase customer satisfaction and employee satisfaction, such a test would only imply that the software adheres to some of the behaviors required by its customer. This is not a limitation of the acceptance testing practice per se, rather it is in the practices that guide the process of acceptance testing. A case in point to this is Miller and Collins (2001), who suggest:

> Your system is done when it is ready for release. It is ready for release when the acceptance tests deemed "must-have" by the customer pass. No other definition makes sense. … What does an acceptance test look like? It says, "Do this to the system and check the results. We expect certain behavior and/or output."

The problem with their approach is that it doesn't provide any guidance on how much testing is enough or how different approaches towards testing

influence the stakeholder values. A common occurrence in the way software practices are designed is that they are influenced by certain guiding principles that have emerged as universal and timeless. Quality has been one such guiding principle that has dominated the last two decades in engineering in forms such as the capability maturity models (Paulk et al., 1994), and total quality management (Crosby 1979; Deming 1986; Juran 1988). Such relentless quest for quality has worked for many organizations, while others have struggled with it.

John Favaro in "When the pursuit of quality destroys value" (1996), observes that "[The way] Quality metrics have been used per se make no explicit strategic or economic statement." He explains that if a bank encourages its loan officers to minimize the percentage of bad loans, its officers will soon discover that by lowering the overall number of loans they can lower the percentage of bad loans. This, however, will bring less money for the bank. While pursuing quality is generally good, it does not substitute for other value drivers.

A Case for Value-Based Engineering of Software Systems

Why are value-based approaches better?

The IEEE Standard defines "software engineering" as a "quantifiable approach to the development, operation, and maintenance of software." A value-based approach to engineering however is not founded on the principles of quantifiable vs. qualitative, right vs. wrong, but builds on Webster's definitions of

"engineering" – and therefore aligned towards "usefulness" (value) to its stakeholders. If the purpose of developing a software system is to make it useful for its stakeholders, then a value-based approach will yield better results than traditional approaches.

Bullock in "Calculating the value of testing" (2000, p. 61) stated:

> Business is what the military euphemistically calls a "target-rich environment." There are always far more opportunities for process improvement (or other investments) than there are resources available. There's never "enough" money to "do it right" for anybody. Businesses are highly interactive systems, in which each function influences the whole in many ways. Nothing is more useless than a function that has been highly optimized in isolation.

His ideas echo the spirit of value-based approaches. Using a value-based lens, developers of software systems address both – the conflicting, competing, and dependent stakeholder values, and the larger system context.

Huang and Boehm's (2006) study on software quality, market share erosion, and risk show how a value-based approach to testing can yield better results than using a traditional approach. For example, they showed (in Figure 2) how value-based testing generated Pareto optimality based on empirical data from Bullock (2000), as opposed to an automated test generation tool that treats all test cases as equally valuable.

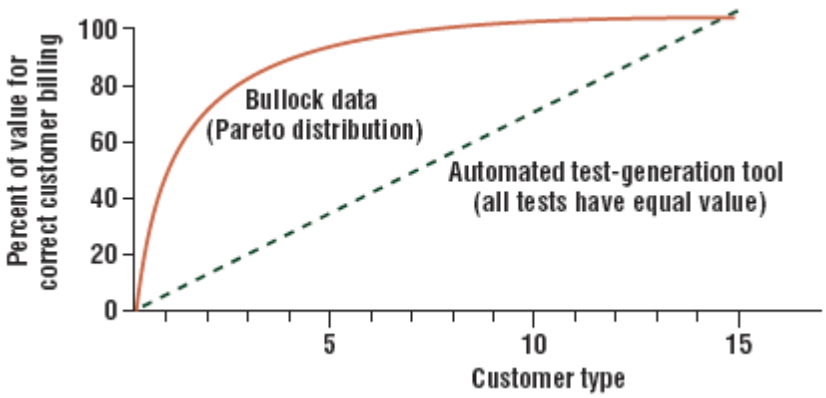
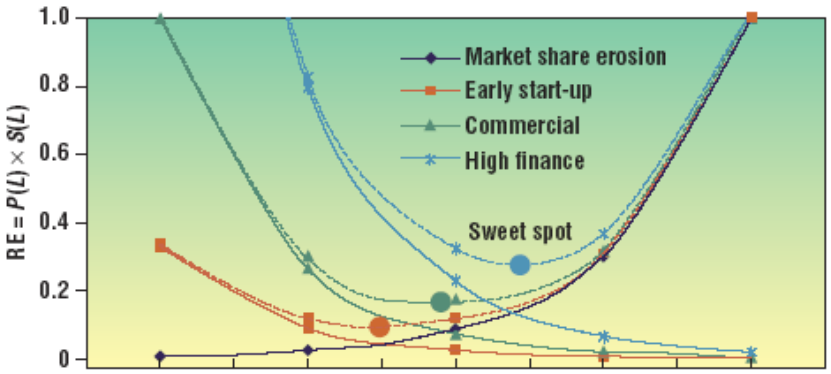


Figure 2. Value-Based vs. Value-Neutral
Source: Huang and Boehm (2006)

Figure 2 describes the testing context for a telecom organization's billing system, which has its services being used by sixteen different types of customers – and with some customer types generating more revenue than the others. Bullock showed how testing each customer type improved billing revenues from 75 to 90 percent and that one of the fifteen customer types generated 50 percent of all billing revenues. If initial testing is focused on that one customer type, it will provide a much quicker and better (in terms of future values) return on testing investment.

In Figure 3 Huang and Boehm (2006) showed how different organizations should reason about their investments on software quality. For example, an early start-up will have a much higher risk impact due to market share erosion than a commercial or a high finance organization. Therefore, from a risk point of view, it is better for an early start-up to field a lower quality product than invest in quality beyond the threshold of negative returns due to market share erosion.



	Very low	Low	Normal	High	Very high	RELY
COCOMO II	0	12	22	34	54	Added % test time
COQUALMO	1.0	0.475	0.24	0.125	0.06	$P_a(L)$
Early start-up	0.33	0.19	0.11	0.06	0.03	$S_a(L)$
Commercial	1.0	0.56	0.32	0.18	0.10	$S_a(L)$
High finance	3.0	1.68	0.96	0.54	0.30	$S_a(L)$
Market risk	0.008	0.27	0.09	0.30	1.0	RE_m

Figure 3. Effect of Software Reliability and Market Share Erosion on Risk
Source: Huang and Boehm (2006)

However, market share erosion is still one in many other factors in an organization's context that must be addressed in value-based testing. By invoking contingency theory, Burton and Obel (2004) show us that an organization's context (see Figure 4) not only includes the market (environment) but also various other elements inside and outside of an organization.

The arrows in Figure 4 indicate empirically demonstrated influences between pair wise elements of an organization's context. For example, if an organization's strategy is based on product leadership then it will have a high risk impact not only from market share erosion but also lack of novelty (Treacy and Wiersema, 1997; Burton and Obel, 2004). Therefore testing practices should also address novelty to be compatible with the organization's strategy. Many

organizations already make use of "usability labs" to get consumer feedback on the product's ease of use, and for the consumers' perception of the product in comparison to other competitor products. Using such approaches, a testing practice should also be able to establish a measure for novelty.

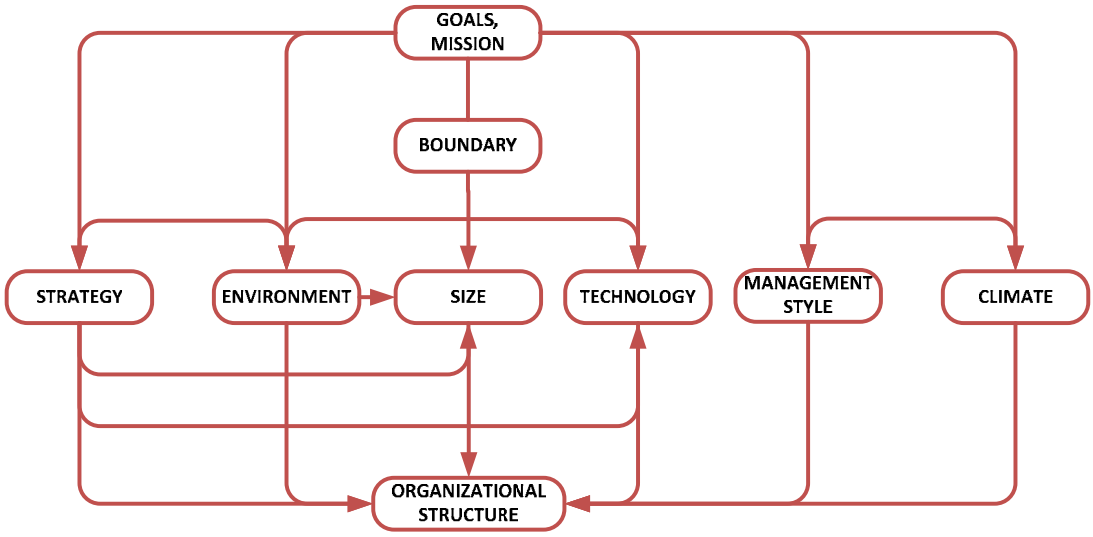


Figure 4. Organization Context
Adapted from Burton and Obel (2004)

Today most practices used in developing software systems have been limited by their unit of analysis – that is, within the boundaries of a project. They do not explicitly include the organizational context when reasoning about a software system. For example, as explained before, a system's level of quality depends on an organizations' strategy, environment etc. (Treacy and Wiersema, 1997; Burton and Obel, 2004; Huang and Boehm, 2006). And, level of quality further depends on the level of testing, available resources, etc. However, most software and systems engineering practices do not address such

interdependencies when providing guidance on, say, software testing. Instead, they only seek to optimize on cost and schedule.

Making decisions about the many competing, conflicting and dependent attributes of a software system has been a complex problem to address. This is because (1) stakeholder values can be boundedly rational and circumstantial, and (2) there are many interdependencies among the software system, involved organizations, and their environments that must be factored into decision making. Fortunately, with application of appropriate methods and theories, these complexities can be addressed.

Conceptual Framework

How does this study address a few of the current engineering shortfalls?

Many engineering research projects focus on tools, techniques, and methods. But in this case that would be premature. As such, the primary research question that this study addresses is:

Can a value-based theory for software engineering be developed that provides:

a. criteria that distinguish projects that will fail from projects that will succeed;
b. a process that applies the criteria to enable projects to succeed and satisfactorily addresses the criteria for a good theory?

The *4+1* Value-Based Theories

How is the theory organized?

A theory that can address the multi-dimensional nature of stakeholder values needs to bring together interdisciplinary theoretical lenses into a state of synchrony and allow reasoning about systems in different dimensions, at different times, and at various levels of abstraction. It therefore needs to address all of the considerations of technical systems engineering theories, plus considerations involved in the managerial aspects of systems, plus the personal, cultural, and economic values involved in developing and evolving successful systems. It also has to have the ability to identify and work through the dependencies of socio-political-technical systems and explain success and failure in such contexts by situating the success-critical stakes at the forefront.

The socio-political-technical nature of systems may not fit well with most of the formalized theories of mathematics and science. Such theories are able to be both formal and predictive because they rest on ideal, universal and timeless assumptions, such as the flow of electricity through a conductor, the flow of air around an airfoil, or the flow of chemicals through a reactor being the same ten years from now as they are now. However, such assumptions of universality and timelessness cannot be made of systems involving people.

Although this sounds complex, using success-critical-stakeholder values to situate and guide technical and managerial decisions actually made this job easier.

The proposed theory presents a set of sufficient conditions for a system to be successful, and a set of sufficient steps (in the next section) for realizing a successful system. Its results depend largely on dealing with the value propositions of the system's success critical stakeholders, and with the interdependencies that their value propositions create. This study lays the foundations for such a theory.

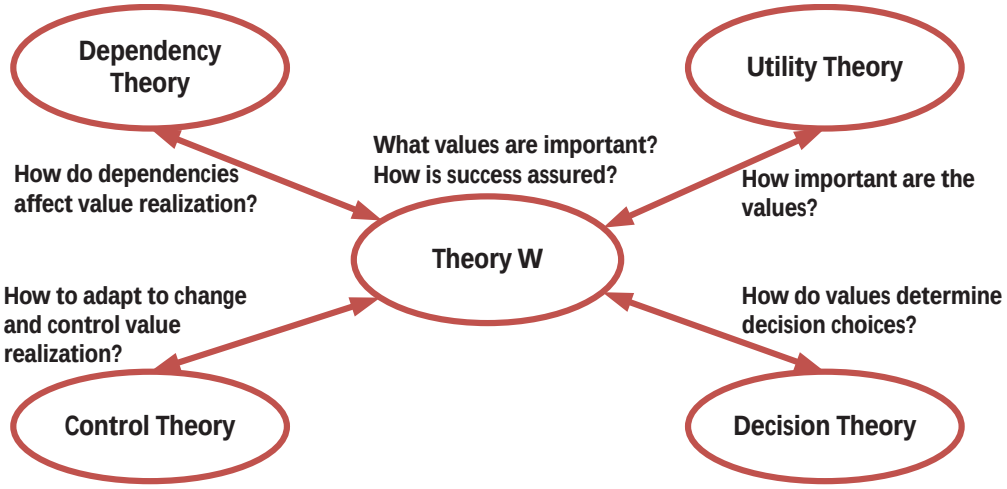


Figure 5. The 4+1 Theory – Key Constructs

Figure 5 is a architectural illustration of the proposed "4+1" theory of VBSE. The engine in the center is the success-critical stakeholder (SCS) win-win Theory W that addresses the questions of "what values are important?" and "how is success assured?" for a given software engineering enterprise. The four additional theories that it draws upon are dependency theory (how do stakeholder, product, and process dependencies affect value realization?), utility

theory (what is the relative importance of each value?), decision theory (how do stakeholders' values determine decisions?), and control theory (how to adapt to change and optimize value realization?).

The Process Framework

How do the theories work together?

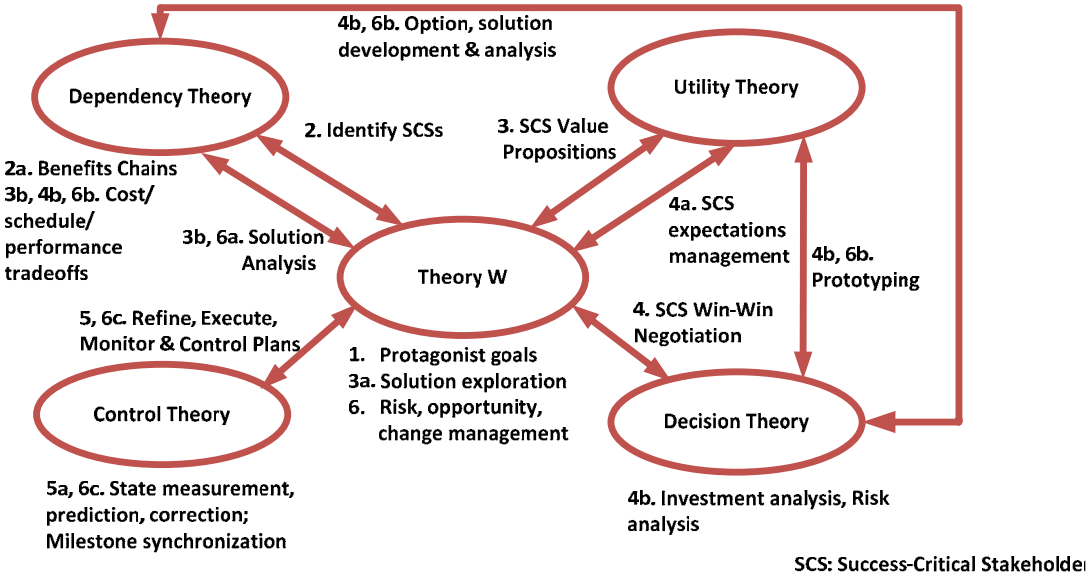


Figure 6. The Process Framework

Figure 6 illustrates the process view of the "4+1" theories. It serves as an enactment guide for its users in six steps.

While it is not a complete step-by-step process for developing a software system, it identifies how traditional engineering practices can be applied in an overarching framework that is driven by stakeholder values. It shows in each step how values can be integrated into the process of developing software systems. For

example, Step 2a, Benefits Chain, connects the software system to its context by uncovering initiatives and assumptions that potentially require attention before significant resources are committed to developing the system.

Significance of Research

What are the primary contributions of this study?

The theory proposed in this study addresses the two identified shortcomings of existing approaches in developing software systems: disconnect between stakeholder values and the values applied during the engineering of a systems product, and timelessness and universality in existing approaches. Many related studies (Huang and Boehm, 2006; Bullock, 2000; Favaro 1996; Thorp 1998) have noted the shortcomings of value-neutrality of existing engineering approaches used in developing software systems. However, this is the "first" theory that has identified the "sufficient and necessary" constructs for practitioners to apply value-based concepts across "all" phases of software systems development life cycle.

Six case studies were used post hoc to assess the strength of the theory and its step-by-step enactment process. The results supported the "sufficient and necessary" criteria for the theory by showing how certain (failure) outcomes could have been avoided if the theory had been applied. The theory and process helped explain success vs. failure by adding stakeholder value, and was particularly

helpful when compared to other theories involving explanations of human behavior. The application in six case studies represents a promising start. This study hopes to strengthen the discipline by contributing a theory that:

- Relies on existing theories: Other methods offer heuristics and biographically-based guidance. This study is based entirely on theory, and, as Karl Lewin so eloquently stated, "There is nothing as practical as a good theory." Theory is thick, generating a rich palette of behavior from a few, parsimonious rules.
- Usability - Other methods for making decisions do not direct the decision-maker in a step-by-step manner. Most other methods try to instill overall, high-level guidance, leaving the decision-maker on his/her own to implement. The 4+1 theory with its enactment guide gives specific instructions in what information to evaluate and how to share information among the steps. And the steps can be implemented at any granularity.
- Empirical validation - Many other methods are dicta, there is no empirical validation at all. The proposed theory rests on a combination of logical analysis (e.g., the theorems and their proofs, which are not as precise as those of mathematics , but are better fitted to human values and qualified with respect to their universality); application and integration of well-established theories; use of quantitative results from research to date in

value-based software engineering; and case studies representing multiple disciplines, sizes, and technology levels.

Limitations

What were the limitations of this research?

This study is limited on two fronts: its validation approach, and overall scope, both as a result of its early stage of evolution. The following sections discuss some of the implications of these limitations, followed by a justification for using such an approach in light of its limitations.

Validation Approach

What were the implications on validity?

Six case studies were identified to apply the theory and test its validity. Although they do not provide general validation, they show consistent results from a variety of domains including public service, finance, office applications, robotics applications, and supply chain management, and consistent results with the complementary analytic results and quantitative results from other value-based software engineering studies.

Representativeness

The selection of case studies was based on content richness, that is, the high degree of detail presented, as opposed to the selection's representativeness

of different project types, technical domains or industries. However still, they covered a variety of domains such as from a variety of domains including public service, finance, office applications, robotics applications, and supply chain management.

Rigor

Rigor in research is normally demonstrated in all of a number of ways. In the case of this study, rigor was observed in resting the theory not on one but on a combination of logical analysis; application and integration of well-established theories; use of quantitative results from research to date in value-based software engineering; and case studies representing multiple disciplines, sizes, and technology levels.

Bias

It is difficult to gauge the extent of bias in the data collected by the authors of the selected case studies, though it would be remarkably coincidental by any measure to assume or assert that the case study authors could bias this research study. On the other hand, there was no bias in the choice of which case studies to select to support the research hypothesis advocated here, as richness of detail and availability were the only criteria used in selecting case studies.

Research Scope

What were the implications on scope?

The scope of this research is modest indeed, as theories pertaining to sociotechnical systems cannot just rely on any one form of validation. However still, this study makes a sincere attempt by combining results from a variety of perspectives; using the strengths of complementary research modes in demonstrating variety and consistency; and including results from existing research conducted in the field.

Chapter 2: Literature Review

This study proposes a new theory for developing software systems. Therefore, a review of the relevant literature must include much of the past and existing studies on the subject of software systems; the subject of organizations since they provide the context for development; the subject of economics since developing software systems are a means for economic pursuits; and the subject of social sciences since software systems interact with people. Unfortunately, a review of such breadth is not practical.

The growth of this (systems and software engineering) field alone in the last five decades has been so rapid that it has caused a literature explosion. The Institute of Electrical and Electronics Engineering (IEEE) today houses over 55,000 qualified (reviewed and accepted by a bona fide committee) publications with the word “software” in a document’s title or abstract; and over 1,000 of which, use the term “software process” in their abstracts. If the Association for Computing Machinery, Wiley, Springer, and other publishers of the many books written on software methodologies were to be included, these numbers will probably be in the order of thousands. Additionally, the fields of organization design, economics, and social sciences, each like software systems have an equally impressive body of knowledge.

This study has been done towards a doctoral dissertation. This significantly limits the time and resources available. As such, this chapter examines relevant literature in the following way. In the first part "Theories of Software Systems" it identifies a subset of the literature that is (a) most prominently used in the field, and (b) relevant to this research in helping distinguish this study from its predecessors. In the second part "Candidate Theories for VBSE," it reviews a very small subset of literature from other fields (organization design, economics, and social sciences) that this study used to construct a new theory, and to show a proof of concept that can be further built upon.

Part I: Theories of Software Systems

Theories of software systems (how best to develop software systems) have ranged from the code-and-fix model in the 1950s and 60s that literally had no theoretical underpinnings to it, to the recent emergence of the agile methodologies that have drawn concepts from chaos theory in the making of more descriptive models for developing systems.

Review of literature related to software systems shows that there are many different theories used in the process of developing software systems. For example, some such theories are limited for cost estimation, some for doing architecture and design, and others for the many other downstream activities that are involved in the course of development.

However, this review is limited to lifecycle models (theories) that provide assistance in (a) guiding the core activities of development and (b) explicitly tries to answer “how best to develop a software system”. The next few sections examine the various lifecycle models that have proliferated in the field of software systems.

The Code-and-Fix Model

Code-and-fix (commonly referred as the “hacking” approach) is perhaps the most criticized model in developing software systems due to its lack of discipline. This is because it starts with little or no initial planning. However, code-and-fix has continued to remain a widely-seen choice for software developers, especially when they are faced with a tight development schedule.

Boehm in “A spiral model for software development and enhancement” (1988) states that code-and-fix was the "basic model used in the earliest days of software development [and] contained two steps: (a) Write some code, (b) Fix the problems in the code. Thus, the order of the steps was to do some coding first and to think about the requirements, design, test, and maintenance later" (pp. 61-62). Thereafter, the 2-step model (code and fix) iterates until either the project is declared complete or cancelled.

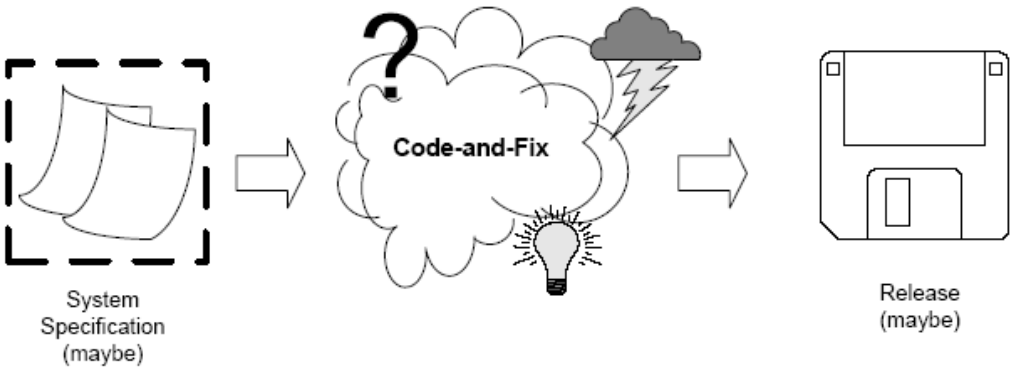


Figure 7. The Code-and-Fix Model
Source: McConnell (1996, p. 140)

There are two significant advantages of using the code-and-fix model. First, it has no overhead in terms of time spent in planning, documentation, and verification and validation. Second, it requires little expertise since it doesn't have any formal requirements for application – anyone who writes software (without using any other formal theories) is using code-and-fix by default (McConnell, 1996).

However, this model has its share of difficulties too – first, since it doesn't require that the stakeholder values be identified and explicated, the resulting software may poorly match the needs of its stakeholders, resulting in outright rejection or rework; second, since it doesn't engage in any upfront design, after a number of fixes, the software usually becomes so unstructured that additional fixes becomes very difficult; third, without any verification and validation planning, the software becomes very expensive to fix in the later phases (Boehm,

1988). In the timeline of theories of software systems, code-and-fix was perhaps the first theory, though granting it theory status may be a stretch; it was certainly an observed practice.

In comparison to this study, the code-and-fix model is both stakeholder value disconnected and context disconnected in its approach.

The Waterfall Model

The waterfall model is a sequential software development model in which development is seen as flowing steadily downwards (like a waterfall) through the phases of requirements analysis, design, implementation, testing (validation), integration, and maintenance (Wikipedia.com). The waterfall model is usually known as the classic model for developing software systems. First documented by Winston W. Royce in 1970, it is perhaps still the most widely used software development process, in one form or another (Humphrey, 1989). The pure waterfall lifecycle consists of several non-overlapping stages, as shown in the Figure 3. The model begins by establishing system requirements and software requirements and continues with architectural design, detailed design, coding, testing, and maintenance. Each phase in the Waterfall model concludes with a review to determine if the project is ready to proceed to the next phase or continue iterating the current phase or going back to previous phases. The

waterfall model today continues to serve as a baseline for many other lifecycle models.

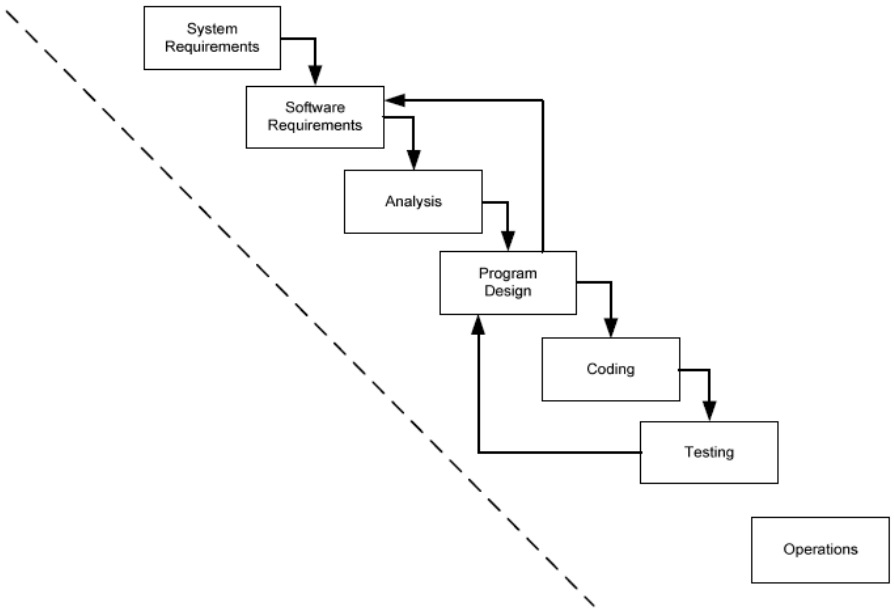


Figure 8. The Waterfall Model
Adapted from (Royce, 1970)

McConnell (1996) argues that the waterfall model works best for projects that have a stable product definition and well-understood technical methodologies, and it works especially well if the project staff is inexperienced since it provides the project with a structure that helps in minimizing rework. Boehm (1988) further explains that the fundamental disadvantage of the waterfall model is that it emphasizes on fully elaborated documents as completion criteria for the requirements and design phases at the beginning of the project. Since in most cases certain requirements and design issues are only uncovered after

construction has begun, substantive modifications in subsequent phases may lead to a poorer design. Finally the Waterfall model requires that a software system's requirements are pre-specified however, when commercial off-the-shelf (COTS) products are employed, system requirements usually become emergent than pre-specifiable.

To address some of the issues in the classic waterfall model, a number of "modified waterfall" models have been proposed in the subsequent years – these modifications have focused on allowing some of the stages to overlap, thus reducing the documentation requirements and the cost of returning to earlier stages to revise them. Another common modification has been to incorporate prototyping into the requirements phases. While overlapping stages, such as the requirements stage and the design stage make it possible to integrate feedback from the design phase into the requirements, they also make it difficult to know when each phase ends, when to proceed to the next phase. Consequently, progress is more difficult characterize and to track. Without distinct stages, problems can cause deferring important decisions until later in the process when they are more expensive to correct.

Although the waterfall model in general has been criticized for its disadvantages, Glass (2003) points out that it is still preferred in certain management circles. The reason is that the waterfall model has the simplicity of explanation and recall, and it gives the sense of control, an "orderly, predictable,

accountable, and process, with simple document-driven milestones, such as requirements complete" (Larman, 2004, p. 106).

In comparison to this study, the Waterfall model to some extent captures the stakeholder values in form of a requirements document. However, stakeholder values, as explained above, usually change, possibly causing costly problems downstream in the lifecycle. Additionally, the Waterfall model does not make an explicit connection between stakeholder values and development activities, such as in architecting or testing. Finally, the Waterfall model is also disconnected from its context.

The Spiral Model

Barry Boehm (1988), in addressing the shortcomings of the Waterfall model, proposed the spiral model as a “risk-driven process model generator that can be used to guide multi-stakeholder concurrent engineering of software-intensive systems. It has two main distinguishing features. One is a cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.”

He showed that with the spiral model’s core emphasis on risk management, major problems can be found much earlier in the development

cycle. For example, in the waterfall model, software design must be completed before construction. With the spiral model, a project is broken down into a set of risks that guide the course of development. For each spiral (or iteration), it requires that the most important risks are analyzed and addressed, only after which the project can proceed to the next issue (that is, risk in order of importance).

According to McConnell (1996) the basic idea of the spiral model is "you start on a small scale in the middle of the spine, explore the risks, make a plan to handle the risks, and commit to an approach for the next iteration," and "after the major risks have all been addressed, the spiral model terminates as a waterfall lifecycle model would" (p. 141).

He explains that the biggest advantage of the Spiral model is that as costs increase, risks decrease, and if the risks are insurmountable, the model will give the project an early indication, which will save time and money. However, the model also has two disadvantages. First, it is complicated, requiring conscientious, attentive, and knowledgeable management (McConnell, 1996); second, in many cases the spiral model is not easily decomposable into the progressively finer levels of details in real-world's software development projects (Humphrey, 1989).

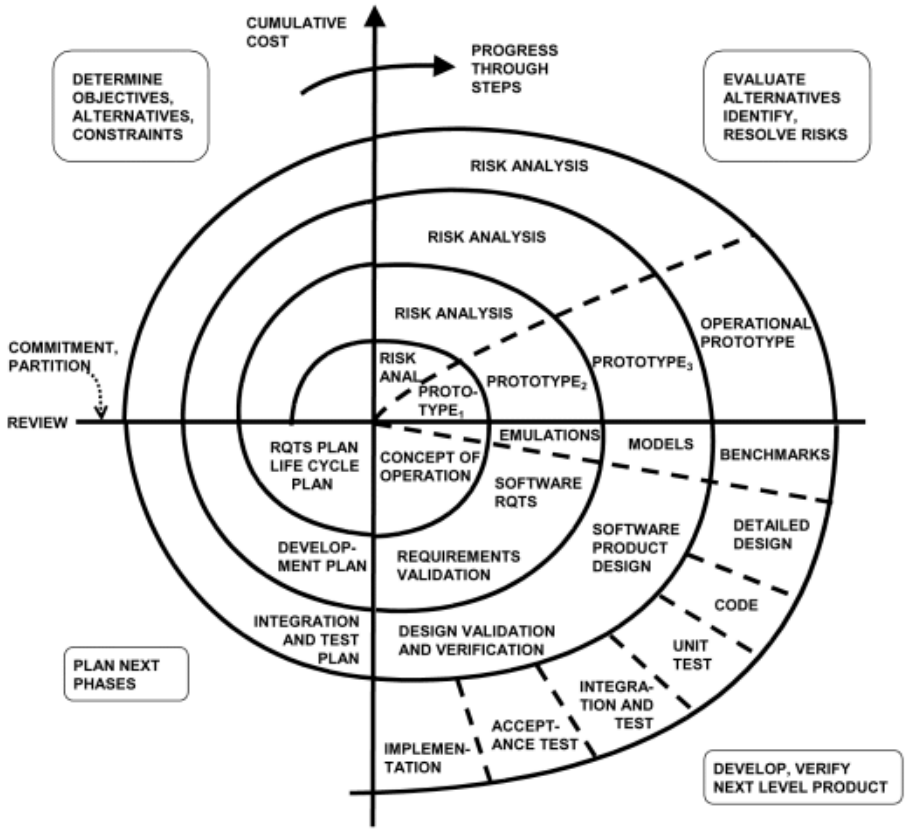


Figure 9. The Spiral Model
Source: Boehm and Hansel (2005)

In comparison to this study, the Spiral model iteratively tries to capture stakeholder values in forms of objectives, constraints and priorities, and risk however, it makes no explicit connection with the organizational context. For example, it does not require that organizational dependencies such as the environment be identified. Finally, it has nothing to say about monitoring project progress with respect to the stakeholder values.

Rapid Prototyping Models

Humphrey (1989) defines rapid prototyping models as those types of processes that aim to reduce the requirement uncertainties through demonstrations of facets of system behavior. According to Weinberg (1991) rapid prototyping enables system developers to create the most prominent parts of a program as a prototype and then work with users to refine it until the prototype is good enough. Once accepted, the prototype either becomes the basis for the final product or is discarded.

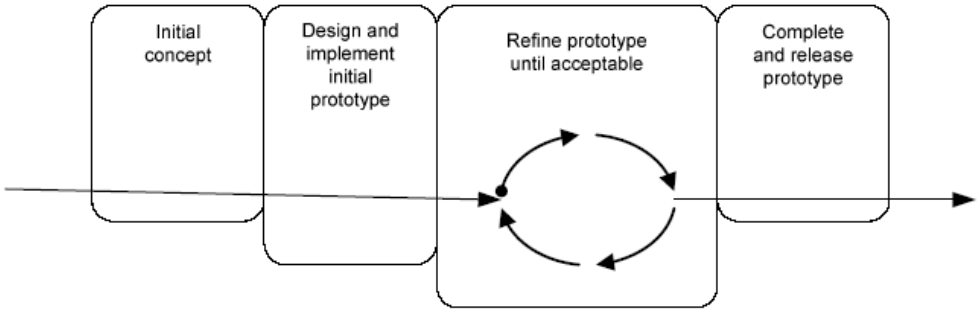


Figure 10. The Rapid Prototyping Model
Source: McConnell (1996, pp. 147)

As mentioned, one of the main problems with the Waterfall model is that the requirements often are not completely understood in the early development stages. Rapid prototype models instead serve as an effective tool for demonstrating how a solution meets a set of requirements. One can build a

prototype, adjust the requirements, and revise the prototype several times until all the stakeholders have a clear picture of the overall solution. In addition to clarifying the requirements, a prototype model also defines many areas of the design simultaneously.

However, McConnell (1996) argues that its strength is also a source of its biggest weakness. Since it may seem to many stakeholders that there exists a working system, they may expect a complete system sooner than is possible. In most cases, prototypes are built on compromises that allow it to come together quickly but prevent the prototype from being an effective basis for future development. Also, using a prototyping model can usually become a disguise for a code-and-fix development cycle. Finally, it makes it literally impossible to make any estimates about the cost or schedule of the project in developing the software system.

In comparison to this study, the rapid prototyping model while tries to capture some stakeholder values through iterative prototyping, it does so at the cost of subordinating many other desirable attributes of a software system. For example, an inherent problem in the rapid prototyping approach is that it does not scale up to large applications, or non-user interface oriented applications. Additionally, the Rapid Prototyping model is also disconnected from its context.

Part II: Candidate Theories for VBSE

As stated in Chapter 1, the primary research question that this study addresses is:

> Is there a sufficient theory for the process of developing software systems that can provide a unified framework for reasoning about how to incorporate stakeholder values into the myriad "decisions" required to deliver such systems?

A decision is a choice given a set of preferences (utilities), and partial information on the causality (dependencies) between choices and outcomes. Therefore, literature relating to "utilities", "dependencies", and frameworks for analyzing "decisions" is discussed. Additionally, in Chapter 1, Figure 4, this study showed how the organizational context has an impact of the way software systems are developed. Since in most cases the organizational context is not static, literature on "control" mechanisms is also reviewed. Finally, this section also examines a subset of literature on "organization theories" to gain a better understanding on how the development process of software systems is influenced by the organizational context.

Organization Theories

Organizations have been studied in many disciplines such as economics, social sciences, psychology and business. Each such discipline has offered many different views and perspectives on how organizations should be studied. To help

understand these various perspectives, some authors have offered frameworks that distinguish among these perspectives based on their assumptions, similarities, historical context, unit of analysis, and epistemological standing. This study uses the Rational, Natural, and Open Systems classification as proposed by Richard Scott in *Organizations: Rational, Natural, and Open Systems* (2003). An alternative framework that is equally prominent is the Functionalist, Interpretive, Radical Humanist and Radical Structuralist classification as discussed in Burrell and Morgan (1972).

According to Scott, three distinct perspectives have been employed in the study of organizations: organizations as rational systems, as natural systems, and as open systems. However, as also acknowledged by Scott, in some cases these distinctions will fade as overlaps (hybrids) occur. Nevertheless, his classification is one based on commonality in ideologies across different theories. To this end, he explains:

> Proponents of organization as a rational systems view organizations as "collectives oriented to the pursuit of relatively specific goals. They are purposeful in the sense that the activities and interactions of participants are coordinated to achieve specific goals. Goals are specific to the extent that they are explicit, are clearly defined, and provide unambiguous criteria for selecting among alternative activities." And, organizations are "collectivities that exhibit a relatively high degree of formalization. The cooperation among participants is conscious and deliberate; the structure of relations is made explicit and can be deliberately constructed and reconstructed ... Formalization refers to the extent that the rules governing behavior are precisely and explicitly formulated and to the extent that roles and role relations are prescribed independently of the personal attributes and relations of individuals occupying positions in the structure."

> Proponents of organization as a natural system view organizations as "collectivities whose participants are pursuing multiple interests, both disparate and common, but who recognize the value of perpetuating the organization as an important resource. The informal structure of relations that develops among participants is more influential in guiding the behavior of participants than is the formal structure".
>
> And, proponents of organization as open systems view organizations as "congeries of interdependent flows and activities linking shifting coalitions of participants embedded in wider material-resource and institutional environments."

Today, there are many different engineering methodologies that are currently used in the development of software systems. Ideally, for process research, it would be helpful to know what view of organization (or at least what organizational characteristics) these engineering methodologies have assumed. However, with a few exceptions, engineering literature usually does not make this explicit. Earlier, distinction between two different approaches used to be based on the characteristics of software systems however, recent literature has shown that the effectiveness of various methods also vary based on organizational characteristics (Borchers, 2003, Boehm and Turner, 2004).

In examining current engineering methodologies used for developing software systems, such as the Waterfall (Royce, 1970), Spiral (Boehm, 1988) and V (Sommerville, 1999) lifecycle models; improvement frameworks such as CMMi (Ahern et al., 2003) and TQM (Crosby 1979; Deming 1986; Juran 1988); and general systems theories such as (Forrester, 1961; Sterman, 2000; Wymore, 1967, Alexander, 1979; Rechtin, 1991; Newnan, 2004; Marschak and Radner, 1972;

Wiest and Levy, 1977); most seemed to fit well with the ideologies of closed systems. However, as mentioned earlier, some of these may have some overlaps.

This study views organizations as open systems. In particular, in the next chapter, this study uses a contingency approach as a form of dependency theory to show how development of software systems depends on the organizations' external environment.

The rest of this chapter will now review some organizational theories that have taken an open systems view (contingency, institutional, and resource dependency) as forms of dependency theory. Following which, utilities, decision analysis, and control mechanisms will be discussed.

Dependency Theory

Dependency theory has its origin in the field of social sciences and refers to a theory that was developed in the 1950s to explain interdependencies between the developed and developing nations. It argued that the less economically developed countries depend on more economically developed countries, and will continue to do so as the formers' surpluses will be siphoned off by more economically developed countries. Studying interdependence between two nations is not relevant to this study. However, the term "Dependency theory" is still used since the primary objective of this theoretical lens is to understand connections.

As conceived and applied in this study, it refers to any set of principles that offers to explain the nature of inter- and intra- dependence between the various elements and aspects of developing software systems (processes, products, people, internal and external environments) that affect value realization. For example, the development approach of a software system will depend on an organization's strategy. If an organization's strategy is oriented towards product leadership, then innovation and time to market become the key drivers in the development approach. On the other hand, a strategy oriented towards operational excellence aligns best with a low-cost product, high-quality product and process focus and innovation.

This section reviews three streams of research: contingency theory, institutional theory, and resource dependency theory. Each of these has offered a set of principles and an explanation for how various organizational elements and entities interact.

Contingency Theory

Often termed as the "it depends theory," contingency theory refers to a stream of research that posits that not all organizations are the same, and thus, a "blanket approach" ("one size fits all," universal and timeless) is not appropriate (Daft, 2003). Daft observes that effective organizations exhibit an appropriate fit between external environmental conditions and their internal organizational

structure. Therefore the correct administrative approach is contingent on the organization's external environment.

Richard Scott (2003) explains that contingency theory challenges the assumptions of those administrative theorists who have sought to develop a set of normative principles for all organizations, in all times and places.

More tersely stated by Jay Galbraith in *Designing Complex Organizations* (1973) and Richard Scott in *Organizations: Rational, Natural, and Open Systems* (2003) contingency theorists believes that:

a) there is no one best way to organize; however,

b) any way of organizing is not equally effective.

c) the best way to organize depends on the nature of environment to which the organization relates.

Contingency theory has its roots in the 1960s and onwards (Burns and Stalker, 1961; Chandler, 1962) when increasing numbers of organizational scholars began to see the impact of factors such as strategy and environment on an organization’s structure. This period observed an increasing trend towards defining the term "environment" and “strategy,” prioritizing key factors, and developing techniques to achieve congruence between organizational characteristics and structure.

In the history of organization design, contingency theory found a distinct space for itself through the promising publications by Joan Woodward (1970), Jay

Lorsch and Paul Lawrence (1972), Richard Daft (2003), Jay Galbraith (1973), Jeffery Pfeffer and Gerald Salancik (1978), and Charles Perrow (1973).

While in the literature of contingency theory there has been an increased emphasis to show the interdependence between an organization's environment and its structure, today contingency theory refers to any number of management theories that subscribe to the "it depends" ideology. Some contingency theorists subscribe to the idea of technology determinism; others have focused on environment uncertainty, strategy, leadership style, and climate as other sources of primary contingencies (Burton and Obel, 2004). For example, Chandler in his study of four companies (du Pont, General Motors, Standard Oil, and Sears Roebuck) found that an organization's structure should depend on an organization's strategy. To this end, he succinctly stated that "Unless structure follows strategy, inefficiency results" (Chandler, 1962 pp. 314-315).

Joan Woodward in her study of 100 organizations in South Essex, Britain, found that the type of tasks that an organization undertakes (manufacturing process and technologies) has a dramatic impact on its structure. According to her, choice of manufacturing process and technologies used by an organization could severely limit the organizational choice of management (Woodward, 1970). Therefore, she argued that bureaucracy would appear to be the best form of structure if organizations are involved in routine operations. However,

decentralization and an emphasis on interpersonal processes will work better in environments of non-routine work.

James Thompson in *Organizations in Action* (1967) observed that organizations operate at three distinct levels: institutional, managerial, and technical. At the highest level (institutional), an organization operates like an open system, interdependent with its external environment. At the lowest level (technical), it operates like a closed system, protected from the organization's external environment. And at the middle, the managerial level mediates the two by providing a buffer between the two end points. Thus, each level is directly interdependent on its upper level, indirectly interdependent on all levels.

Coining the term "contingency theory" is attributed to Jay Lorsch and Paul Lawrence (1972). They argued that different environments in the uncertainty-certainty continuum place different requirements on organizations. An environment characterized by uncertainty and rapid change presents a different set of opportunities and constraints for an organization and its subunits than a placid and stable environment. In their study of chemical industries (plastics, food processing, and standardized containers) they also showed that different subunits within an organization are exposed to different environments, and thus pose different challenges for the various subunits. For example, they showed that research and development units in a plastics manufacturing firm will usually be exposed to higher levels of uncertainty and rapid change than production.

However, this may not be the case for other types of industries. The primary hypothesis of their study was that the more varied the types of environment confronted by an organization, the more differentiated its internal structure needs to be. And, the differentiation and mode of integration characterizing the larger organization should be aligned to the overall complexity in the environment in which the organization must operate (Scott, 2003).

Jay Galbraith's framework of contingency theory (1973) is founded on the principles of information management and processing. In *Organization Design* (1977) he states that "the greater the task uncertainty, the greater the amount of information that must be processed among decision makers during task execution in order to achieve a given level of performance". Therefore the primary challenge for organizations is to organize themselves around their needs of information processing. According to him, there are five key design levers that an organization can use to align itself with its information processing needs. These include environmental management, creation of slack resources; creation of self-contained tasks for reducing information processing needs; and creation of lateral relations and vertical information systems for increasing the organization's information processing capacity.

Burns and Stalker (1961) in their study of twenty different organizations found that organizations when exposed to new and unfamiliar environmental demands (such as novel market situations), best align with an organic

management style and structures. However, a mechanistic system is suitable when environmental conditions are stable. Therefore, they, like other contingency theorists, concluded that there is no universal set of principles for neither successful organizations nor an ideal type of management style. The critical management task is to interpret the market and technological situation in terms of its stability or instability, in order to design the appropriate organizational structure for successful implementation.

To conclude, Lex Donaldson in *The Contingency Theory of Organizations* (2001) discusses three limitations of contingency theory. The first problem relates to the seemingly static nature of contingency theory because it appears to discuss change only as a movement from misfit into equilibrium. However, organizations can frequently move in and out of an equilibrium situation. Organizations can experience repeated increases of change in contingencies and organizational structure resulting in a more dynamic theory. The second problem is the difficulty managers have in knowing exactly what organizational structures fit their contingencies. Donaldson proposes that a full fit with their environments is unrealistic for most organizations and suggests the term "quasi-fit" in referring to the partial fit of an organization with its environment. The third problem relates to the idea of the fit line being one of iso-performance, the position that produces equal organizational performance. However, this raises the question of what is the

benefit in becoming a more fit organization if the additional costs are greater than the rewards involved.

Resource Dependency Theory

How external environments impact organizations and their response to these external constrictions was the focus of Jeffrey Pfeffer and Gerald Salancik's book, *The External Control of Organizations* (1978). Pfeffer and Salancik intended their book to be a guide for designing and managing organizations that are externally constrained. It declares that concepts such as organizations and environments have not been readily accepted in the realm of organizational management. However, the basis of their book is that the environmental context of the organization must be comprehended in order to understand the behavior of a company. Organizations are inescapably linked with the circumstances of their environment. The authors' position clearly echoes the tenets of contingency theory. They believe that organizations survive to the extent that they are effective in their adaptation to acquire and maintain resources in environmental change.

What happens in an organization is a consequence of the environment (Pfeffer & Salancik, 1978). There are three environment levels that Pfeffer and Salancik describe in their book. The first level is the entire system of interconnected individuals and organizations. The second level is the combination of people and organizations with whom the organization directly interacts. The

third level is the organization's perceptions and representation of the environment. The authors draw on the work of F. Emery and E. Trist who describe environments to consist of four types (Pfeffer and Salancik, 1978). The first type refers to a situation in which the resources desired by organizations are randomly distributed throughout the environment called a placid-randomized environment. The interconnection between the different elements in the environment is not strong. Organizations can survive as individual and small units.

The placid-clustered is the second environment type in which the pattern of resources is sequentially predictable. It is profitable for organizations to understand the wider environment and its opportunities. A need to formulate plans that will permit an organization to achieve specific objectives is critical. Planning and the development of specific competencies are encouraged in this type of environment and lead to larger, more hierarchical organizations.

The third environment type refers to as a disturbed-reactive environment. In this environment organizations must seriously consider competitive strategies, concentrate on its resources, and organize their efforts in terms of a distinctive plan where they can outwit their competitors. The final type is similar to the third except a much higher level of interconnection occurs among environmental actors. This environment is referred to as the turbulent field and is characterized by an increasing unpredictability of a competitor's actions. This unique

environment has rarely been seen in world history and requires original forms of strategic action.

Institutional Theory

Developed in the 1970s, institutional theory emerged from the methodological and epistemological struggle among the various schools of thought in organization design. According to Scott (2003), institutional theory challenged ideas that economics could be reduced to a set of universal laws and assumptions. Additionally, institutional theory believes that individual and organizational actions are affected by institutions and habitual relations to others (Scott 1995). Although continuing to challenge mainstream economics by examining transaction costs and market failures (DiMaggio and Powell 1991; Scott 1995), institutional theory focuses on issues of legitimacy and constraint caused by structures and reduced agency (Perrow, 1979; DiMaggio and Powell, 1991).

Institutional theory argues that organizational activity and action cannot be explained purely by considering the rational activity of managers (Goodrick and Salancik 1996). The general theme of institutional theory is that organizations operate in highly normative contexts, and organizational survival and behavior is driven more by the need to conform to norms of acceptable behavior and legitimacy than by rationality or efficiency arguments (Meyer and Rowan 1977; DiMaggio and Powell 1983; Scott 1987). Primarily concerned with regulative, cognitive, and normative perspectives that provide meaning and stability to social

life (Scott 1994), institutional theory considers that organizations are mainly concerned with survival and reducing uncertainty (DiMaggio 1988; Scott 1995). Regulative elements of institutional theory explain conformity through sanctions; cognitive elements explain beliefs and taken-for-granted assumptions such as bounded rationality; and normative elements explain traditional and social obligations.

Like contingency theory, institutional theory has many variations across the different works of its proponents. However, this study is limited to new institutional theory as described in (Meyer and Rowan, 1977; DiMaggio and Powell, 1991). In sum, the key ideas explained by new institutional theory are organizational homogeneity, the stability of institutionalized components, legitimacy and the embeddedness of organizational fields, and cognitive processes where normative obligations are imposed upon actors (DiMaggio and Powell 1983; Scott 1995). In a word, organizations are seen via this lens to imitate each other, which is why we do not see in the field all alternatives possible, why there are, for example, only a few different life cycles choices explained and used.

Utility, Decision, and Dependencies

As stated earlier, decisions are choices given a set of preferences (utilities), and partial information on the causality (dependencies) between choices and outcomes.

If there are no choices, there is nothing to decide. Therefore, the first prerequisite of decision analysis is that there is more than one alternative available to a decision maker. The second prerequisite requires that the preferences of people involved can be identified. If there are no preferences, then the decision-maker will be indifferent among the outcomes – as such, there will no way to differentiate the preferability among alternative outcomes.

Utilities

Utilities are stated as preferences and indifferences. For example, we might all agree that we prefer more money to less. And that we have a higher preference for the first dollar than to the second, the second to the third, etc. (diminishing marginal returns). University professors prefer that term papers are turned in by the end of the term, and after that are (comparatively) indifferent about when they are turned in.

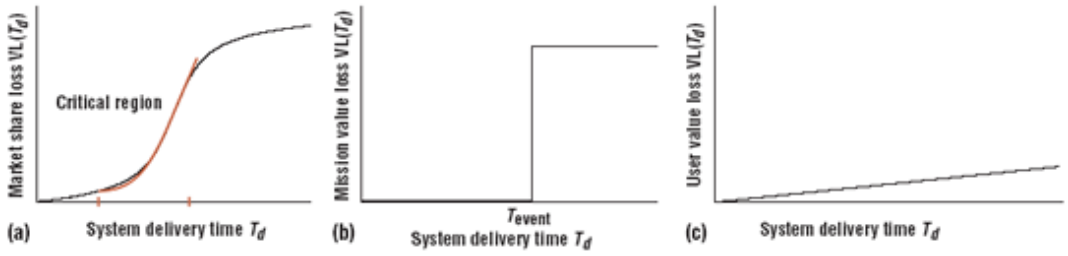


Figure 11. Stakeholder Utility Functions
Source: Huang and Boehm (2006)

Utility is usually presented graphically with thing being valued (time to market, profit, and usability) along the abscissa and the corresponding preference along the ordinate. Huang and Boehm (2006) show the following examples of utility relationships:

Maslow's Hierarchy of Needs as a Utility Theory

Abraham Maslow in *Theory of Human Motivation* (1943) identified five fundamental human needs and their hierarchical relationship. A key aspect of his study was in establishing the hierarchical nature of human needs. His model in Figure 12 shows that the lower the needs in the hierarchy, the more fundamental they are and the more a person will tend to abandon the higher needs in order to pay attention to sufficiently meeting the lower needs.

Figure 12. Maslow's Hierarchy of Needs

Maslow's model was not proposed as an economic theory of utility, but he did try to explain how people make decisions by identifying what they need so his model can be considered as a description of utility.

Decision Theories

Defined as the "act of making choices" (Keeney, 1988) or a "commitment to action" (Mintzberg et al., 1976), utility and decision theories have been primarily conceptualized as rational theories of choice, use the subjective expected utility (SEU) model (March and Heath, 1994), and operate within the assumptions of bounded rationality (Simon, 1957). The rest of the section examines decision (and perforce utility) theories using two different frameworks, as conceptualized by (Mintzberg et al., 1976) and (Bell et al., 1988), and then later examines other forms of decision and utility theories.

The Bargaining, Judgment, and Analysis Framework

The first framework for characterizing decision analysis is to consider it as a process of bargaining, judgment, and analysis (Mintzberg et al., 1976). The bargaining process involves making the many trade-offs among multiple decision options with conflicting goals (negotiating in a politically motivated situation). The judgment process is an individual approach where decision makers select an option using a process that they may not consciously recognize. And analysis is a discovery process where alternatives are objectively evaluated with respect to a set of goals.

Of these three arms of the framework, analysis has been the most emphasized in normative literature (Mintzberg, 2004) and is most frequently manifested by subjective expected utility. First proposed by von Neumann and Morgensten (1944), SEU is a formal utilitarian model, where decision makers choose by assigning probabilities to possible states of nature and utilities to outcomes, calculate expected utility (probability of the state of nature times the utility of the outcome given that state), and then select the choice option with the highest expected utility. Today, SEU in the form of multiple-attribute decision making has a widespread adoption across many fields, despite questions about its assumptions, usefulness, and applicability (Arrow, 1988; Bell et al., 1988).

The Descriptive, Normative, Prescriptive Framework

The second framework for understanding decision analysis is to identify the extent to which it is descriptive, normative, or prescriptive (Bell et al., 1988). Descriptive decision research focuses on decisions people make and how people actually decide. Normative research emphasizes logically consistent decision procedures and how people should decide. Prescriptive research deals with how to help people make good decisions and how to train people make better ones.

This framework does not make a distinction among the desirability of each approach. Each is ideal for a specific (research or business) goal.

Rationality

Rationality, which is "accorded the methodological privileges of a self-evident truth" (Tversky and Kahneman, 1988), today remains a central aspect of decision analysis. As Harsanyi (1966, p. 139) notes:

> From the point of view of a social scientist trying to explain and predict human behavior, the concept of rationality is important mainly because, if a person acts rationally, his behavior can be fully explained in terms of the goals he is trying to achieve.

The term rationality, however, is often imprecisely defined. Beyond the accepted and widely held belief that there are limits on human cognitive capability (Simon, 1957), rationality is frequently used in at least two different ways. Action can be considered "rational or substantively rational" if it results in outcomes that are deemed good, and "procedurally rational" if made by a procedure that

assesses expected consequences and chooses actions that are expected on average to lead to desired outcomes. Under these conditions, processes may be judged rational even if the outcomes are not consistent with expected results – rationality procedures may lead to good outcomes or poor outcomes. In addition, rationality of the decision cannot be determined *ex ante*; it exists strictly as an *ex post* concept, and many decisions that are judged intelligent may be subsequently assessed as unintelligent when their outcomes and effects on values are weighted (March and Heath, 1994).

Intuition, Symbols, Ceremony and Rituals

Beyond the normative dominance of rationality in decision analysis there are also other descriptive aspects of decision analysis such as intuition, symbols, ceremony, and rituals. Although frequently neglected (March and Heath, 1994) these are important elements of decision analysis because they provide meanings for actions.

Intuition has been referred to as the tapping of an unstructured and indirectly accessible lifetime accumulation of experiences (Simon, 1957). Symbols, rituals, and myths are to varying degrees independent and frequently intertwined (Pettigrew, 1979), but they can still be differentiated in definition. A symbol is an object, practice, or sign that evokes something else by association, and links an organization's experience to deep feelings or to abstract definitions of human dilemmas. A myth is usually a fictional story that appeals to the consciousness of

people by embodying its cultural ideas or by giving expressions to deep, commonly felt emotions; it establishes what is legitimate and unacceptable in an organizational culture (Pettigrew, 1979). A ritual is a set of ceremonial forms or the symbolic use of bodily movement and gesture in social situations to express and articulate meaning, or by a mechanism by which the traditions are preserved and meanings sustained (March and Heath 1994).

The Garbage Can Model

The garbage can model (Cohen et al., 1972) is a descriptive approach to decision analysis that attempts to explain organizational decision making anomalies (not being normative). In particular, it explains decision making by "organized anarchies" where preferences are not clear, technology is not clear, and/or participation is fluid.

The primary thesis of the garbage can model is that an organization "is a collection of choices looking for problems, issues and feelings looking for decision situations in which they might be aired, solutions looking for issues to which they might be the answer, and decision makers looking for work" (Cohen et al., 1972). They explain that problems, solutions, and decision makers move from one choice to another depending on the mix of recognized problems, choices, and available choices for problems. Organizational problems are addressed based on a choice of a solution, but choices are made by engineering combinations of problems, solutions, and decision makers. Poorly understood and addressed problems drift

into and out of the garbage can process, depending on the situation and factors. Therefore, decision making in organizations are not always rational in that they do not follow a rational process.

The spirit of decision theory is that a good decision (choice) will result in a good outcome. In the context of this study, a good decision is an informed decision – one that explicates the causality between the different choices available and their likely outcomes, and identifies the best choice that will lead to a good outcome. A good outcome is one that realizes the desired values of a software system's stakeholders, such as profitability or system performance. Good decisions may not always lead to good outcomes however, without true clairvoyance available to a decision maker there is no better alternative in the pursuit of good outcomes than to make good decisions.

Control Theory

The central tenet of control theory is that dynamic systems are deployed in a changing environment that may cause the system to be instable. Therefore, applying control mechanisms to a system during operations can enhance the value of such dynamic systems.

According to Blanchard and Fabrycky (1998), an explicit concern for control was witnessed in the 1950s when system engineers turned their attention to large-scale, human-made systems, in which, there were many states, control

variables, and constraints. They explain that while classical control theory as in mechanical systems cannot be used directly to select measures of effectiveness and to optimize outputs, nevertheless, concepts can still be used to provide a basis for structuring systems and internal relationships (dependencies) so that feedback and adaptive phenomenon are incorporated in (software) system design (and in management as used in this study).

The key aspect of control theory relevant to this study is the system controller and its two mechanisms feedback and feedforward. They are discussed next.

Feedback Control

A feedback control mechanism refers to a control system in which the output of a system is monitored through a sensor (continuously or at some intervals) and then processed through a control device that compares actual vs. planned performance. If the actual performance does not match planned performance, corrections are made through the actuator, and then the input is "fed back" into the system.

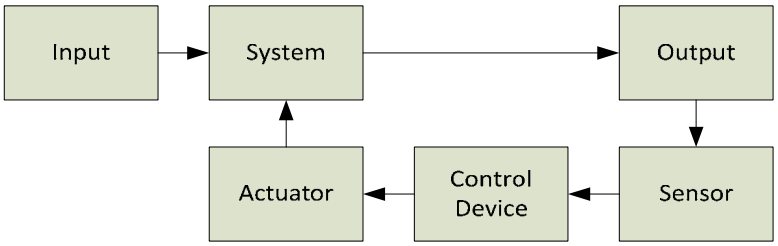


Figure 13. A Feedback Control System

Feedforward Control

Feedforward control, in contrast to a feedback control does not have an interrupt in the system for corrections and decision making if actual performance does not match planned performance. However, correction can still be made as long as the algorithms that will do the decision making are prespecifiable, and any external changes sensed by the sensors are measurable.

Implementing either of the two control mechanisms makes some assumptions of the nature and state of the environment. As summarized in (Brogan, 1974) these include observability (the ability to observe the current enterprise state), predictability (the ability to predict whether the enterprise is heading toward an unacceptable state), controllability (the ability to redirect the enterprise toward an acceptable near-term state and a successful end state), and stability (the avoidance of positive feedback cycles that cause control systems to overcompensate and become unstable).

Chapter 3: To Theory

Karl Lewin said, "There is nothing as practical as a good theory." Theory is thick and it generates a rich palette of behavior from a few, parsimonious rules. With this aspiration, this chapter presents the theory of value-based software engineering, with an emphasis on the key theoretical constructs, and on the interactions between these constructs.

Definitions and Context

This section traces the context for using existing definitions or constructing new ones that will be used through the rest of the chapter, and elsewhere in the study.

Success

> Mahoney in “Finding a History for Software Engineering” (2004) says:
>
> Dating from the first international conference on the topic in October 1968, software engineering just turned thirty-five [now thirty seven]. It has all the hallmarks of an established discipline: societies (or sub-societies), journals, textbooks and curricula, even research institutes. It would seem ready to have a history. Yet, a closer look at the field raises the question of just what the subject of the history would be … What is a history of software engineering about? Is it about the engineering of software? If so, by what criteria or model of engineering? Is it engineering as applied science? If so, what science is being applied and what is its history? Is it about engineering as project management? Is it engineering by analogy to one of the established fields of engineering? If so, which fields, and what are the terms of the analogy? Of what history would the history of software engineering be a part, that is, in what larger historical context

> does it most appropriately fit? Is it part of the history of engineering? The history of business and management? The history of the professions and of professionalization? The history of the disciplines and their formation? If several or all of these are appropriate, then what aspects of the history of software engineering fit where?

Much of Mahoney's epistemological questions still remain unanswered. One explanation for this is that if history speaks of successes and failures, then the field is yet to be blessed with a universally acceptable definition for both success and failure. With every new theory, there is an intrinsic view of success and failure on which the theory is based. For example, for researchers inspired by science, success is achieved only through following a set of laws and rules. They have sought mathematical formalism and completeness in their theories, and revered predictability (Jones, 1980). Similarly, for researchers inspired by behavioral sciences, success is dependent on the alignment of human and skill factors (Juristo et al., 2004).

This study is not an exception. It has also conceived a view of success, and it is imperative that it is explicated to allow making any distinction between this study and its predecessors. As stated earlier in Chapter 1, p. 1, this study considers the activity of developing software systems as a means to an end for the people who directly or indirectly depend on it. Having taken such a view thus implies that success of a software system is to the extent that it delivers the expected benefits to its stakeholders. Therefore, using this definition, a view of "maximized" success (as in the theorems of Theory W) is that all success-critical stakeholders become

winners. Since, success in most practical situations cannot be measured in "absolute" terms; this study uses the concepts of "satisficing" and "success-criticality" as discussed in the Choice Rational section.

Theory

As stated in Chapter 1, the socio-political-technical nature of systems doesn't fit well with most of the highly formalized theories of mathematics and science. These are able to be both formal and predictive because they rest on strong universal and timeless assumptions, such as the flow of electricity through a conductor, the flow of air around an airfoil, or the flow of chemicals through a reactor being the same ten years from now as they are now. However, such assumptions cannot be made about universality and timelessness of systems involving people and continually changing software.

There are numerous definitions of "theory" to consider. They range from highly formal definitions such as, "A theory is a system of general laws that are spatially and temporally unrestricted and nonaccidental" (Hempel and Oppenheim, 1960; Danto and Morgenbesser, 1960), to relatively informal definitions such as "A theory is any coherent description or explanation of observed or experienced phenomena" (Gioia and Pitre, 1990). To capture the strengths of both formal and informal approaches, this study uses Torraco's (1997) definition of "theory".

> "A theory is a system for explaining a set of phenomena that specifies the key concepts that are operative in the phenomena and the laws that relate the concepts to each other." (Torraco, 1997)

Similarly, this study's theorems about success criteria for software-intensive enterprises will not be "spatially and temporally unrestricted and nonaccidental". Systems and their success are subject to multiple concurrent influences, some of which are unpredictable. For example, a project that is poorly requirements-engineered and architected, poorly managed, behind schedule, and over budget can still turn into a great success with the appearance of just the right new COTS product to satisfy stakeholder needs. The reverse is true as well: "The best laid plans o' mice an' men Gang aft agley" through unforeseeable external circumstances (Burns, 1785).

Choice Rationale

To restate, this study addresses:

> Is there a sufficient theory for the process of developing software systems that can provide a unified framework for reasoning about how to incorporate stakeholder values into the myriad decisions required to deliver such systems?

In chapter 1, a case was made about why stakeholder values must be incorporated in the various decisions that are made in the development of software systems. In chapter 2, many different theories most relevant to this study's research question were reviewed with respect to decisions, dependencies,

control, and organizations. In particular, review of utilities and decision theories showed how preferences (utilities) can be incorporated into decision analysis. The focus of this section is to identify the key theoretical constructs required in such a theory, and provide a choice rationale for the resulting set.

Historical Context

To this end, it will be helpful to shed a light on the historical context of this study. The concept of value-based engineering of software systems was first conceptualized and introduced to me by my advisor Barry Boehm. While to both of us it seemed certain that value-based approaches were significantly superior over most traditional approaches for developing software systems, it was not clear what the theoretical underpinnings of such an approach were.

In an earlier study by Boehm and Ross in "Theory W Software Project Management Principles and Examples" (1989), they had proposed that the success of a software system depends on the various success-critical stakeholders of a software system, and making them winners was necessary for success. They further explained that software engineering was not a zero-sum game, and in most cases cooperation will achieve better results than competition.

Since the foundation of this study is to align decisions about software systems such that they deliver to the values expected by its stakeholders, Theory W's primary proposition (make winners out of success-critical stakeholders)

became the “maximized” goal for my theory. As such, Theory W was not only included as a key theoretical construct but its theorems also served as a guiding philosophy that helped in identifying the other constructs.

Theory W

Two theorems form the core of Theory W, the “Fundamental System Success Theorem” and the “System Success Realization Theorem.” The first theorem “Fundamental System Success Theorem” states that:

> “A system will succeed if and only if it makes winners of its success-critical stakeholders.”

As mentioned in Chapter 1, and earlier in this section, value-based theorems and proofs are less formal than those in such areas as mathematics and physics. As such, an informal proof follows.

Proof of “if”:

(a) Everyone significant is a winner.

(b) Nobody significant is left to complain.

Proof of “only if”:

(a) Nobody wants to lose.

(b) Prospective losers will refuse to participate, or will counterattack.

(c) The usual result is lose-lose.

The proof of “if” is reasonably clear. The proof of “only if” may not be so clear, so it is further illustrated in three frequently-occurring examples of the

primary stakeholders in an enterprise involving a customer contracting with a developer for a software system that will benefit a community of users, as shown in Table 2.

Table 2. Win-lose Generally Becomes Lose-Lose

Proposed Solution	"Winner"	Loser
Quick, Cheap, Sloppy Product	Developer & Customer	User
Lots of "bells and whistles"	Developer & User	Customer
Driving too hard a bargain	Customer & User	Developer

In Case 1, the customer and developer attempt to win at the expense of the user by skimping on effort and quality. When presented with the product, the user refuses to use it, leaving everyone a loser with respect to their expectations.

In Case 2, the developer and user attempt to win at the expense of the customer (usually on a cost-plus contract) by adding numerous low-value "bells and whistles" to the product. When the customer's budget is exhausted without a resulting value-adding product, again everyone is a loser with respect to their expectations.

In Case 3, the user and customer compile an ambitious set of features to be developed and pressure competing developers to bid low or lose the competition. Once on contract, the surviving bidder will usually counterattack by colluding with the user or customer to convert the project into Case 2 (adding user bells and whistles with funded Engineering Change Proposals) or Case 1 (often by

exploiting ambiguities in the contract Statement of Work to deliver a contractually compliant but unusable system that must either be thrown away or expensively fixed) Again, everyone is a loser with respect to their expectations.

The Fundamental System Success leads to the System Success Realization Theorem that identifies the predicates of making winners of a system's success-critical stakeholders. The "System Success Realization Theorem" states that:

> Making winners of your success-critical stakeholders (SCSs) requires:
>
> (a) Identifying all of the SCSs.
>
> (b) Understanding how the SCSs want to win.
>
> (c) Having the SCSs negotiate a win-win set of product and process plans.
>
> (d) Controlling progress toward SCS win-win realization, including adaptation to change.

Theory W as a Choice Rationale

The first predicate is self explanatory – "Identifying all of the SCSs" involves the use dependency theory. Chapter 1 showed how Benefits chain analysis, a form of dependency theory can help in associating various stakeholders across different initiatives.

The second predicate "Understanding how the SCSs want to win" requires both dependency theory to identify the dependencies underlying their value propositions (win conditions), and utility theory to identify their value

propositions (win conditions). Chapter 2, in Figure 11, showed how stakeholder values could be converted into utility functions.

The third predicate, "Having the SCSs negotiate a win-win set of product and process plans" requires a combination of utility theory (Chapter 2, in Figure 11) in situations when their value propositions need to be adjusted, dependency theory for understanding the interdependencies (tradeoffs) involved in negotiations (as in Chapter 1, Figure 1 and Figure 4), and decision theory (as explained in Chapter 2) for evaluating the different options against the various value propositions.

The fourth predicate "Controlling progress toward SCS win-win realization, including adaptation to change" involves the use of control theory. As explained in Chapter 2, software systems are developed and deployed in a changing environment that may have significant consequences towards making the system instable, as such control mechanisms can help in controlling and adapting to change.

The four predicates of the System Success Realization thus (in form of requirements) provided a guideline in identifying the other key theoretical constructs of the final theory.

Research Problem as a Choice Rationale

This section presents another rationale for the identifying the key constructs by relating the research problem to the definition of decision analysis. The research problem that this study addresses is:

> Is there a sufficient theory for the process of developing software systems that can provide a unified framework for reasoning about how to incorporate stakeholder values into the myriad "decisions" required to deliver such systems?

As evident in the research problem, this study is addressing how to incorporate stakeholder values into the myriad "decisions" required to deliver such systems. Thus, by the virtue of the research problem, decision theory must be identified as a key theoretical construct in the resulting theory. As defined earlier in Chapter 2, a decision is a choice given a set of preferences (utilities), and partial information on the causality (dependencies) between choices and outcomes. Per this definition, utility theory and dependency theory are prerequisites of applying decision theory. Therefore, both are valid theoretical constructs for the resulting theory. Additionally, as stated in Chapter 2, this study, and in general the concept of value-based engineering of software systems takes an open systems view. Since in an open system, environment is not considered static, forms of control must be applied to prevent any instability resulting from change. Therefore, choice of control theory is also valid.

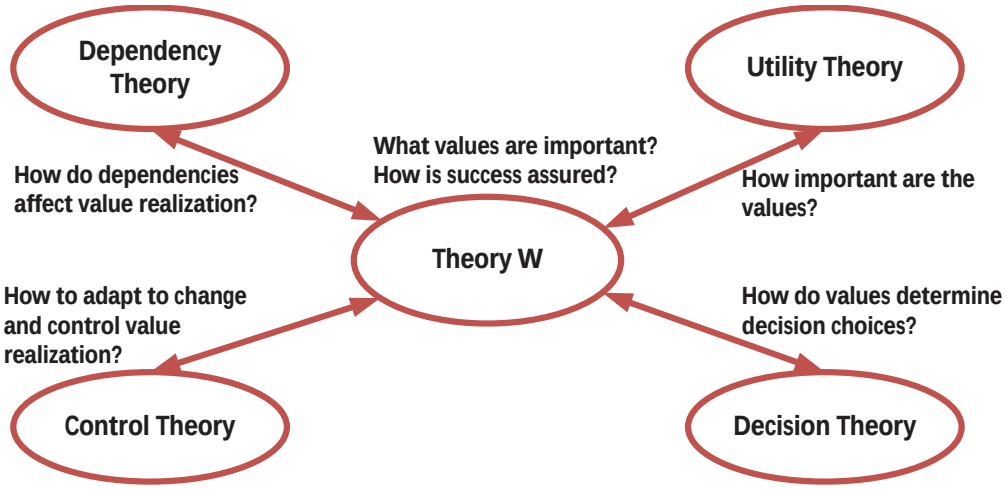


Figure 14. The 4+1 Theory -- Key Constructs
Also Figure 5

Figure 14 summarizes the "4+1" structure of the VBTSE. The engine in the center is the success-critical stakeholder (SCS) win-win Theory W, which addresses the questions of "what values are important?" and "how is success assured?" for a given systems engineering enterprise. The four additional theories that it draws upon are dependency theory (how do dependencies affect value realization? On what stakeholders does success depend), utility theory (how important are the values?), decision theory (how do stakeholders' values determine decisions?), and control theory (how to adapt to change and control value realization?).

Dependency theory

Suppose that a system's original requirements specify a one-second response-time for a large order-processing system. However, the best available COTS database management systems only provide a two-second response time for the system's expected workload. In most cases such a one-second response time will be re-evaluated to match the best available COTS product. Building one's own database management server would be the other option, but usually infeasible due to budget and schedule constraints (it's not a requirement if you can't afford it). As such, a dependency among a system's response time requirements, what a COTS product offers, and the time and cost constraints is established.

We should care about dependencies because they can help decision-makers:

(a) Make better decisions by aligning system decisions with stakeholder values and the organizational context;

(b) Illuminate conflicting stakeholder value propositions between before they turn into serious problems;

(c) Uncover factors that are external to the system and can impact the project's outcome.

In the following sections this study shows how different forms of dependency theories can be used in the course of developing a software system.

Huang and Boehm's (2006) value-based software quality model and Burton and Obel's (2004) multi-dimensional contingency model is used to show how system decisions can be aligned with stakeholder values and the organizational context; and Al-Said's (2004) model clash framework is used for identifying stakeholder value conflicts. Some other significant systems engineering works that address people-product-process dependencies are Warfield's Science of Generic Design (Warfield, 1995) and Checkland's Soft Systems Methodology (Checkland, 1999).

Aligning System Decisions

If a project manager asks "How much testing is enough before I field my system?" VBTSE's response would be "it depends". A natural follow-up would then be "On what does it depend"? VBTSE's response would be on the stakeholder values, and on the organizational context. For example, (Huang and Boehm, 2006) provides a value-based software quality model that helps us answer this question. They show in Figure 3 how different organizations should reason about their investments on testing. For example, an early start-up will have a much higher risk impact due to market share erosion than a commercial or a high finance organization. Therefore, from a risk point of view, it is better for an early start-up to field a lower quality product than invest on quality beyond the threshold of negative returns due to market share erosion. In their model, consequently, they have established a dependency between a system's quality, the organization's business model, and value functions relating expected market

share to product introduction date. And, they show how system decisions such as testing can be aligned such that the project decisions are in alignment with the stakeholder values in the start-up situation.

Market share erosion, however, is one of many dependencies in an organization's context that must be addressed in decision-making. By invoking contingency theory (Burton and Obel, 2004) show us that an organization's context (see Figure 4) not only includes the market (environment) but also various other elements of an organization. The arrows in the figure indicate empirically demonstrated dependencies pair wise between the various elements of an organization's context.

For example, if an organization's strategy is based on product leadership then the project will have a high risk impact not only from market share erosion but also from lack of novelty (Treacy and Wiersema, 1997; Burton and Obel, 2004). Therefore system decisions should be made such that they are aligned with the overall organizational strategy. For example, many organizations already make use of "usability labs" to get consumer feedback on the product's ease of use, and for the consumers' perception of the product in comparison to other competitor products. This helps them establish a measure for novelty and innovation.

In Figure 15, this study further combines the multi-dimensional contingency model (Burton and Obel, 2004) with the (Boehm, 1989) process decision table to show an extended view of project dependencies. Figure 15

shows that system decisions should also include a project's objectives, constraints and priorities (in the Systems column) such that system decisions are also aligned with its stakeholders' value propositions and other system-level constraints.

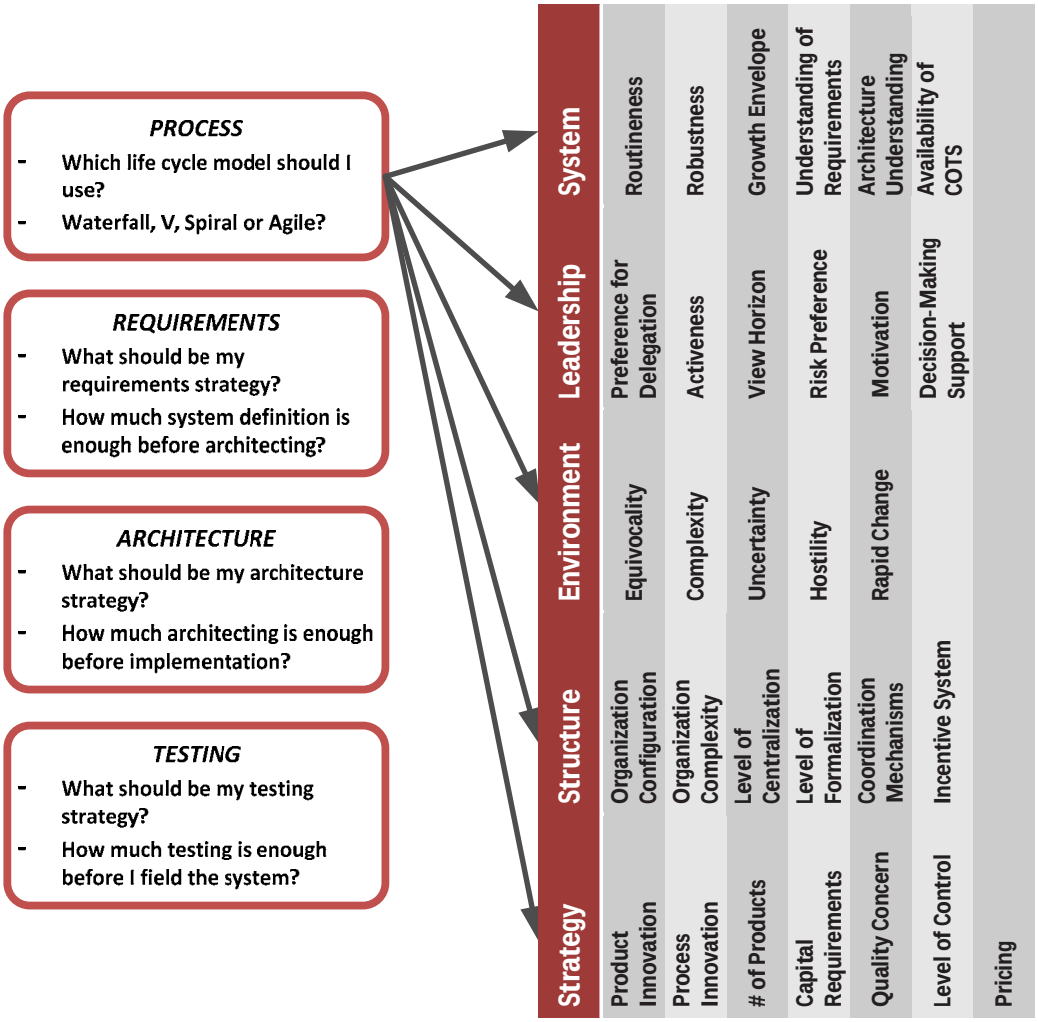


Figure 15. A Combined Dependency Model

For example, when reasoning about the choice of a life cycle model, (Boehm, 1989) shows (in the Systems column) that when a system's growth envelope is "limited to large"; understanding of requirements, need for robustness, and architecture understanding are "high"; then using the Waterfall model is usually the best fit. At the organizational level, (Burton and Obel, 2004) further show that choice of a life cycle model (they use the term technology) also depends on the environment's uncertainty – if there is a high uncertainty in the environment then a routine technology (Waterfall Model) will conflict with environment. Further, organizational strategies on product and process innovation, leadership styles should also be aligned with the choice of process models. For example, if the level of control is high, preference for delegation is low, and preference for risk-aversion is high, then agile methods such as Extreme Programming will conflict with the organization because agile methods are optimized for change and uncertainty.

Today most practices used in developing software systems are either value-neutral, or are limited in their unit of analysis – that is, within the boundaries of a project. They do not explicitly include the stakeholder values or the organizational context when reasoning about a software system. Instead, they only optimize on cost and schedule. Using some of the approached described above, we have shown that identifying such dependencies and aligning system

level decisions with the many other elements in Figure 5 is critical to the success of a project.

Conflicting Stakeholder Value Propositions

A dependency between two elements is established if they interact with each other. In a dependency relationship two elements could conflict, compete or with complement each other. The (Al-Said, 2003) study of model clashes (see Figure 16) show how different stakeholder value propositions can conflict with a system's choice of process, product and property models through their underlying assumptions. The four quadrants in the figure identify the most common project stakeholders (users, acquirers, developers and maintainers) and the most common value propositions they usually have. While each line identifies a conflict between these value propositions, the light gray represent the model clashes that plagued Bank of America's MasterNet project (Boehm et al, 2000).

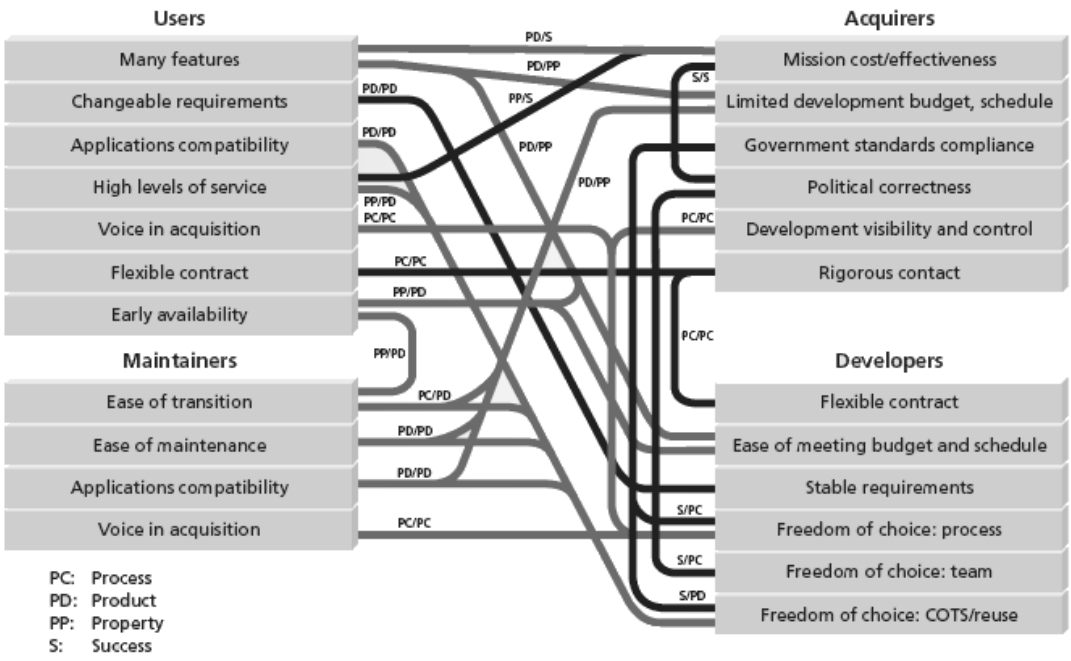


Figure 16. Stakeholder Value Conflicts
Adapted from (Al-Said, 2003)

For example, users want many features, such as freedom to redefine the feature set at any time, compatibility between the new system and their existing systems, and so on. However, they show that the users' value propositions conflict with other stakeholders' value propositions. For example, the users' "many features" conflicts with the acquirers' "limited development budget and schedule" and with the developers' "ease of meeting budget and schedule."

Using forms of dependency theory is not new to the field of systems engineering – literature within the engineering field addresses them with terms such as model clashes, mismatches, conflicts, incompatibility, interoperability, and

at various levels of abstraction and stages such as system definition, requirements, design and analysis, implementation and test traceability and consistency. There are potentially a very large number of dependencies that can exist in a system. Some help in connecting – internally or externally. However, not all are equally important. Some dependencies, when not identified can have a significant impact on the project's outcome, whereas others can be identified and resolved in the course of development. The risk exposure approach exemplified in Figure 3 is a good framework for prioritizing the dependencies that most significantly contribute to the project's outcome. There are many models in addition to the ones described above that can be used to identify and resolve a project's success-critical dependencies. In the following four sections, we provide some guidance and reference to such models based on the different dimensions of a project.

The People Dimension

People dependencies are typically rooted in the socio-political dynamics of an organization. Like environment, these are usually exogenous but also endogenous, visible in management hierarchies, inter-entity and intra-entity coordination mechanisms. While people dependencies are usually considered to be "sticky" because of their socio-political nature, the process of identifying all of the success-critical stakeholders (including influencing organizations) early into the project and using techniques such as expectations management and building a

shared vision through win-win negotiations can significantly reduce the complexity of these dependencies. There is a strong body of knowledge on identifying and understanding dependencies in the people dimensions – the most prominent ones are found in sociology and organization theories (Parsons, 1977; March and Simon, 1958; Argyris, 1978; Rifkin, 2004; Daft, 2003); theories about human systems integration, particularly the integration of macroergonomic and microergonomic concerns (Booher, 2003); and theories about how people and initiatives combine to realize successful systems, such as in value chains (Baldwin et al., 2000) and results chains (Thorp, 1998).

The Process Dimension

Process dependencies are fairly pervasive and originate almost from all the dimensions – from environmental dependencies, (for example, compliance to the US Sarbanes-Oxley Act of 2002 required organizations to implement processes that would provide added accountability and visibility into the projects), or from product dependencies (for example, intensive use of COTS products requires processes that allow stakeholder value-propositions to be more emergent through COTS evaluation than pre-specifiable), or sometimes from people dependencies (for example, involving the sales personnel in the development of a supply-chain management system will also require tailoring the development process to include them as success-critical stakeholders). In addition to the literature referred to in the above sections, PERT/Critical Path Methods (Wiest and Levy,

1977), system dynamics (Forrester, 1961), and network flow theory (Ford and Fulkerson, 1962) are also some good approaches in identifying process dependencies.

The Product Dimension

Product dependencies are usually the biggest source of dependencies for a given systems and software project. As with process dependencies, they are influenced by all the four other (environment, people, process, and property) dimensions. However, with the advent of COTS and other technology alternatives available today, they also have strong influences on the process and property dimensions. For example, choice of a system's architecture and architectural components will directly depend on the environment, people, processes used, and budget/schedule/performance properties. Conversely, a COTS-intensive system will also influence the process and properties. Traditional systems engineering theory such as (Wymore, 1967) provide good coverage of product dependencies. For identifying software and system architectural dependencies, (Alexander, 1979; Rechtin, 1991; Shaw and Garlan, 1996; Clements et al., 2003) are excellent sources.

The Property Dimension

Property dependencies deal with the "ilities" of a system and have a unique role in the dependency network as cross-cutting influences. As seen in

Figure 5, acquirers of a system are usually more interested in properties such as cost, schedule, ROI, and value-addition; users in performance, usability, safety, and privacy; developers in cost, schedule, and reusability; etc. Property dependencies as such directly relate to the value-propositions of the stakeholders and thereby the project's bottom line. They also frequently drive the nature of product models (Clements et al., 2002) or process models, such as for safety-critical systems. Overall property characterizations and influences are addressed in (Boehm et al., 1973) and (Chung et al., 1999). For engineering economics, (Newnan, 2004; Marschak and Radner, 1972; Boehm 1981) are a few excellent sources for understanding and dealing with property dependencies.

Utility Theory

Utility theory helps in understanding the preferences (utilities) of a software system's success-critical stakeholders. Misunderstanding SCS utility functions does not guarantee failure if an enterprise happens to get lucky. But, understanding how the SCSs want to win is essentially a necessary condition for WinWin achievement. Utility theory also has several branches, such as the satisficing criterion of bounded rationality (Simon, 1957), multi-attribute utility theory (Keeney-Raiffa, 1976), and its situation-dependent aspects such as the Maslow needs hierarchy (Maslow, 1954) stating that lower-level needs (food and

drink; safety and security) have dominant utilities when unsatisfied and negligible utilities when satisfied.

Dependency theory aids utility theory by providing stakeholders with assessments of the dependencies and tradeoff relationships among their value propositions, including assessments of the relative costs and benefits of options to reduce risk by buying information on the state of nature via prototyping, modeling, simulation, etc.

Decision Theory

Decision theory helps in having a software system's SCSs negotiate win-win plans. As explained in Chapter 2, the theory has many perspectives, such as normative, descriptive, and prescriptive, and different application techniques such as in (Raiffa, 1982; Fisher and Ury, 1981; von Neumann and Morgenstern, 1944; Luce and Raiffa, 1957; Keeney and Raiffa, 1976; Blackwell and Girshick, 1954; Luehrman, 1998). This study takes a normative approach by being explicitly requiring that stakeholders decide on a process and a product plan based on the values of all the success-critical stakeholders.

Control Theory

Control theory helps in controlling progress toward stakeholder value realization by adapting to external changes. As stated in Chapter 2, the necessary conditions for successful enterprise control are observability (the ability to

observe the current enterprise state), predictability (the ability to predict whether the enterprise is heading toward an unacceptable state), controllability (the ability to redirect a project developing a software system towards an acceptable near-term state and a successful end state), and stability (the avoidance of positive feedback cycles that cause control systems to overcompensate and become unstable).

Particularly for the VBSE theory, it is more important to apply control theory principles to the expected value being realized by the project rather than just to project progress with respect to plans. Traditional "earned value" systems have their uses, but they need to be complemented by business-value and mission-value achievement monitoring and control systems as discussed in (Boehm and Huang, 2003). These involve the use of risk management; adaptive control functions such as market watch and plan renegotiation; and multi-criteria control mechanisms such as BTOPP (Scott Morton, 1991; Thorp, 1998) and balanced scorecards (Kaplan and Norton, 1996). Particularly in an era of increasing rates of change, this makes both traditional and adaptive control (Highsmith, 2000) necessary conditions for system success in terms of the System Success Realization Theorem. Dependency theory aids control theory by identifying which stakeholders and artifacts are affected by sources of change, and by assessing cost, schedule, and performance implications of alternative responses to change requests.

Chapter 4: To Practice

This chapter illustrates how the theory is operationalized in practice.

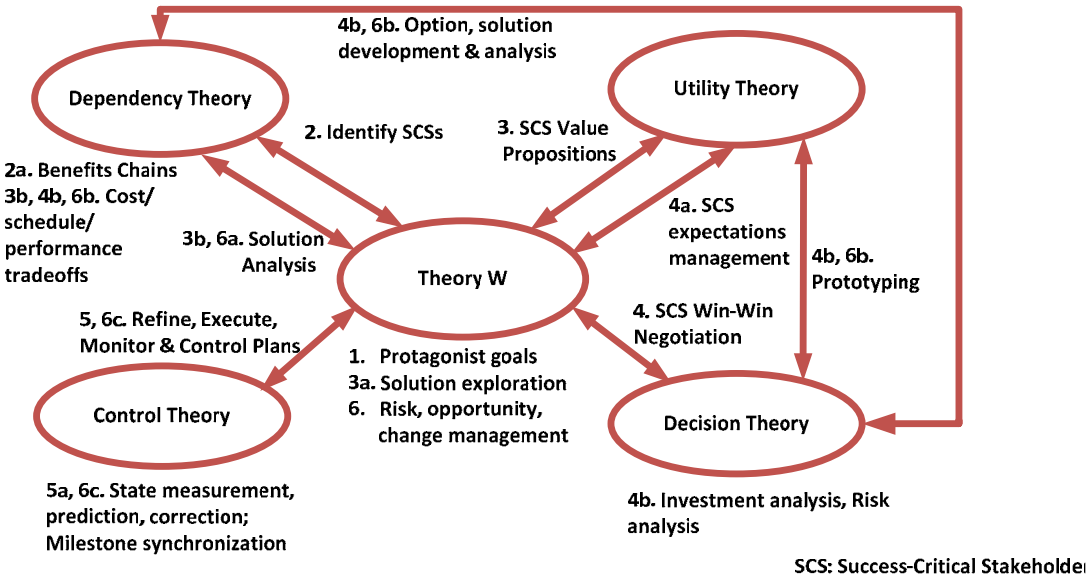


Figure 17. The Process Framework
Also Figure 6

Step 1

Many process models do not address how the system life-cycle process gets started. With VBSE theory, the process gets started when a protagonist becomes convinced that existing systems are seriously inadequate, and that improving them involves connections with/to other success-critical stakeholders. Step 1 of the process identifies the protagonist or change agent who provides the motivating force to get a new project, initiative, or enterprise started. As shown in

Table 3, protagonists can be organization leaders with goals, authority, and resources, entrepreneurs with goals and resources, inventors with goals and ideas, or consortia with shared goals and distributed leadership and resources.

Table 3. Frequent Protagonist Classes

Protagonist Class	Goals	Authority	Ideas	Resources
Leader with Goals, Baseline Agenda	X	X	X	X
Leader with Goals, Open Agenda	X	X		X
Entrepreneur with Goals, Baseline Agenda	X		X	X
Entrepreneur with Goals, Open	X			X
Inventor with Goals, Ideas	X		X	
Consortium with Shared Goals	X	(X)		(X)

Step 2

Step 2 takes a software systems project to the province of dependency theory with particular emphasis on environment, organizations, and people dimensions. In this step, the unit of analysis is people and organizations, and their affiliations with respect to the environment through techniques such as DMR's Results Chain Analysis (Thorp, 1998), social-network analysis, ethnographic analysis, macro-ergonomic analysis, and context-of-use analysis (Booher, 2003). For example, applying results chain analysis to a supply-chain organization would identify success-critical stakeholders such as the suppliers, distributors, retailers,

users, regulatory bodies that control the standards of the products manufactured; along with their relationship to the environment; position in the value chain; and assumptions about the cooperativeness or adversity of these stakeholders and with respect to the market place (competitors and potential new entrants).

Step 3

Step 3 basically involves identifying the utilities (stakeholder values) of all success-critical stakeholders, but also includes some initial solution exploration and cost, schedule, performance analyses. In this step, our unit of analysis spreads through the product and property dimensions and using stakeholder value propositions, a first-cut dependency network is created. In addition to common value propositions of internal stakeholders, Steps 2 and 3 together also often identify product dependencies from external stakeholders such as the distributors, suppliers and retailers requiring compatibility with their existing systems – generating additional sources of product dependencies.

Steps 4, 5 and 6

Steps 4 and 6 are similar in their approaches to identifying dependencies, negotiating with stakeholders, doing business analyses, identifying decision options and establishing control parameters – they are applied in different phases of the project. Step 4 is done before the implementation phase (until the

development team has committed to a single architecture and established its feasibility) and Step 6 is after the project has moved forward to the implementation phase. Step 6's need and rationale is based on the emergent nature of requirements (people dependencies), high rate of technology change (product dependencies), and market volatility (environment dependencies).

In Steps 4 and 6, the process framework basically gets into an iterative loop that continues until a win-win equilibrium is achieved and all stakeholder value propositions have been satisfied. The loop begins with some expectations management, progresses forward through stakeholder win-win negotiations, business analyses and in the process continues to uncover additional product-process-property dependencies by way of prototyping, architecture property tradeoff analysis, and limited-resource (cost, schedule, key personnel, etc.) tradeoff analysis; using risk management techniques such as buying information, risk avoidance, risk transfer, risk reduction, and risk acceptance.

A note on product dependencies and expectations management in Step 4 must be made. As mentioned earlier, product dependencies are by far the largest source of interoperability problems and a primary reason for high costs in building and maintaining next generation systems. For example, stakeholder value propositions such as users wanting compatibility with their existing products, and thereby requiring new systems to integrate with legacy systems has been a very common source of product dependencies. Also with continued evolution of COTS,

having to integrate different COTS products amongst each other and then with the legacy systems, a project often commits to undesired low-flex, low-freedom product dependencies. Identification and resolution of such product dependencies early into the project helps discover a solution space into regions such as feasible, not-feasible, or somewhat-feasible regions based on their dependencies. Step 4 in anticipation of these product dependencies and sometimes others as well, requires an upfront expectations management with all success-critical stakeholders paving the way for a successful win-win negotiation.

If an organization has used steps 1-4 to identify dependencies, determine their value propositions, and develop business cases, it has developed the framework for Step 5 – to monitor expected value realization, adjust plans, and control progress toward real SCS value achievement. As with the dependency analyses, project planning, executing, monitoring, adapting, and controlling in Step 5 proceed incrementally in increasing amounts of detail, generally following a risk-driven spiral process. Questions such as "how much is enough planning, specifying, prototyping, COTS evaluation, business case analysis, architecting, documenting, verifying, validating etc.?" are best resolved by balancing the risk exposures of doing too little or too much. As Risk Exposure = Probability (Loss) * Value (Loss) is a value-based concept, risk balancing is integral to the theory.

Chapter 5: Results

This chapter discusses the results of applying the theory and the process framework on six case studies. The questionnaire used for reviewing each case was derived from an analysis framework built from the process framework and the "4+1" theories as in Table 4. While this chapter only briefly discusses each of the cases to put the many derived insights in context, further discussion on each of the them are provided in Appendices A through E.

Table 4. Analysis Framework

"4+1" Theories	Cause	Explanation
UTILITY	Project fails to identify its success-critical stakeholders and their value propositions	Potentially incomplete or incorrect requirements leading to a product that stakeholders will not approve for further development or release. (example: the sales staff reject a newly acquired order-entry and processing system because it does not track sales credits)
UTILITY	Project fails to prioritize requirements	Suboptimal resource planning and allocation may lead to resource crunch (example: 3 months was spent implementing feature A which was not even important for the release)
UTILITY	Project does not manage expectations	Any deviations from expected results may lead to noncooperation (example: 3 seconds response time is unsuccessful if 1 second is promised, but successful if 5 seconds is promised)
UTILITY	Project does not engage in prototyping	Incorrect requirements leading to stakeholders rejecting the system (example: display information overload)

Table 4. Continued

"4+1" Theories	Cause	Explanation
DEPENDENCY	Project fails to identify its organizational dependencies	Changes in organization can significantly affect the project (a recent corporate restructuring effort impacted the product's design, takes away resources)
DEPENDENCY	Project fails to identify interdependencies between stakeholder values	Usually breaks the win-win equilibrium leading to adverse consequences (example: marketing staff puts in a request for feature X, this increases response time affecting operations staff, loser will fight using the system)
DEPENDENCY	Project fails to identify product interdependencies	Leads to significant design changes, rework (example: architectural mismatch, such as feature A uses SOAP and interfaces with external vendor product using REST)
DEPENDENCY	Project fails to identify process interdependencies	Suboptimal resource planning and allocation may lead to delays and overruns (example: infrastructure software not developed in advance, delaying integration and testing)
W	Project fails to negotiate a win-win product/process plan	Usually breaks the win-win equilibrium leading to adverse consequences (example: marketing staff puts in a request for feature X, this increases response time affecting operations staff, loser will fight using the system)
W	Project fails to use risk to establish feasibility and commitment	Misguided or wrong decisions, lack of support from stakeholders, destabilizes project continuity (example: a key COTS vendor discontinued support, leaving the project without a backup plan)
DECISION	Project does not have a business case	Misguided decisions (example: expended $1M on feature A for which ROI was only 1.2 times over feature B that may have had 2.6 times return on investment)

Table 4. Continued

"4+1" Theories	Cause	Explanation
DECISION	Project fails to use stakeholder values for its product or process related decisions (initially or during changes)	Breaks win-win; leads to misguided decisions (example: expended $1M on feature A for which ROI was only 1.2 over feature B that may have had 2.6)
CONTROL	Project fails to establish a control mechanism to track progress	Decreased visibility into project's progress (example: feature A takes 3 months leaving only 3 months for features B, C and D where were equally critical)
CONTROL	Project fails to monitor and measure stakeholder value	Decreased visibility into project's success (example: system implemented all the features in the requirements document but stakeholders still not happy)
CONTROL	Project fails to adapt its plans to key changes in SCS value propositions	Increases technical and financial risk; breaks win-win (example: expended $1M on a system that clearly had missed its market window halfway through development)

Table 5. Summary of Results

Theory	Process	Assessment (Impact when Negative L/M/H)				
		MasterNet	UniWord	Word	LAS	CMU-SAR
UTILITY	Project identified its success-critical stakeholders and their value propositions	YES	NO (H)	YES	SOME (H)	YES
	Project prioritized requirements	NO (H)	NO (H)	NO (H)	NO (H)	YES
	Project managed expectations	NO (M)	NO (M)	N/A	NO	YES
	Project engaged in prototyping	NO (H)	NO (H)	SOME (M)	NO (H)	YES
DEPENDENCY	Project identified its organizational dependencies	SOME (M)	N/A	YES	NO (L)	NO (H)
	Project identified interdependencies between stakeholder values	NO (H)	NO (H)	SOME (L)	N/A	NO (H)
	Project identified product interdependencies	YES	NO (M)	NO (M)	N/A	YES
	Project identified process interdependencies	SOME (M)	NO (M)	YES	SOME (M)	YES
W	Project negotiated a win-win product/process plan	YES	NO (H)	YES	NO (H)	YES
	Project used risk to establish feasibility and commitment	YES	NO (H)	YES	NO (H)	YES
DECISION	Project had a business case	YES	N/A	YES	YES	YES
	Project used stakeholder values for its product related decisions (initially or during changes)	N/A	NO (H)	YES	NO (H)	YES
	Project used stakeholder values for its process related decisions (initially or during changes)	N/A	NO (H)	YES	NO (H)	YES
CONTROL	Project established a control mechanism to track progress	NO (M)	NO (M)	YES	SOME (M)	YES
	Project monitored and measured stakeholder value	NO (M)	NO (M)	YES	NO (M)	N/A
	Project adapted its plans to key changes in SCS value propositions	SOME (H)	NO (H)	YES	NO (H)	NO (H)

Table 5 provides a summary of the analysis conducted on the first five case studies. As mentioned in Chapter 1, it is the purpose of this research to assess if a value-based theory for software engineering can be developed that provides (a) criteria that distinguish projects that will fail from projects that will succeed and (b) a process that applies the criteria to enable projects to succeed and satisfactorily addresses the criteria for a good theory. As such, the goal of the analysis across the first five case studies was to assess the theory and its derived process to gauge its strength in identifying success from failure through post-hoc analyses. In the sixth case study, the goal shifts to case (b) in applying a theory derived process and evaluating the outcomes through a criteria discussed later in this chapter and eventually in the next chapter.

Case 1: UniWord

Background

This case study was published in (Boehm, 1981). Universal Micros, Inc. was developing a new product: An advanced personal computer named UniWindow. In order to make its product more appealing to its potential customers, Universal Micros decided to contract with several software houses to produce software packages for its new personal computer. One such package was the UniWord – a Word Processor for UniWindow that would be developed by the winning bidder to

the Request for Proposal that was based on a set of high-level specifications provided by Universal Micros.

SoftWizards, Inc., a software contractor, was selected as the winning bidder among four other companies that had also bid for the contract. With some past experiences with developing text editors, Soft Wizards believed that winning this contract would provide a significant value addition to the company. Therefore, in excitement of this prospective opportunity, SoftWizards agreed to build the product with a much lower price tag than its competitors.

Within a few months into the project, problems began overshadowing success. Toward the expected completion date, SoftWizards failed to deliver the product. This initiated a project audit by Universal Micros, which forms the basis of this post-hoc analysis.

Analysis Summary

As evident in

Table 5, the UniWord project scored very poorly on the theory's scorecard. The project overlooked many critical "must-dos" as prescribed by the theory, and therefore the results were strongly indicative of its eventual failure. However still, few observations made during the analysis are discussed next.

UniWord as a project failed to identify many key success-critical stakeholders and their value-propositions, did not have any risk management or control instruments plans guiding the project, and overlooked multiple opportunities in the course of the project to establish a win-win across all stakeholders. The analysis eventually concluded that while UniWord performed very poorly across all dimensions of the theory, having committed to an extremely risky project (unrealistic schedule, little discretionary funds, and stakeholder utilities tied to a fixed date based on an upcoming event) without establishing a risk management plan perhaps had the strongest impact in contributing towards failure.

Case 2: BofA MasterNet

Background

This case study was published in (Glass, 1998). In 1982, Bank of America, along with Seattle-First National, United Virginia, and Philadelphia National initiated the development of a state-of the-art trust accounting system. Having internally failed once in doing so and losing $6 million, the consortium of banks contracted

Premier Systems to develop their trust accounting system that would manage their $38 billion portfolio in institutional and personal trusts.

Premier Systems was a relatively new company but its leadership, primarily Stephen Katz, had a strong background in financial systems and some success developing such systems through his previous company. With some initial research into the new system that the consortium of banks required, Premier began developing the banks' trust accounting system and promised delivery within 11 months.

After spending $78 million on development and losses incurred due to the new system's malfunctions, the banks finally handed their trust business to a subsidiary on account of inability to handle trust related services. Along with financial losses, plenty of bad press such as "$80 Million MIS Disaster" in newspaper headlines, the bank's image as a successful technological leader was negatively changed forever.

Analysis Summary

Bank of America's MasterNet initiative was a huge undertaking by the bank. If successful, the project would have brought significant benefits to the bank's trust services in terms of customer retention, and attracting new customers. Therefore, it had a very strong business case but like the UniWord project it faltered in attending to the many risks that entailed such a project, some

attributed to size and complexity and some through the multiple criss-crossing interdependencies that encompass the project.

In the course of analysis, it was found that overseers of MasterNet faired very well on a few elements of the theory's scorecard – such as involving the various success-critical stakeholders, establishing a win-win product and process plan, and distributing some of the project risks across other banks by forming a consortium of banks interested in the developed system. However still, a project as large as MasterNet not only carries a high risk but also brings many complexities in building and deploying such a system. When faced with such an enormously complex system, the theory absolutely requires that the project engage in expectations management, value and requirements prioritization, and a phased approach derived from such prioritizations. Additionally, the theory finds it imperative that strong instruments for monitoring and controlling progress are put in place to avoid vicious cycles of rework and have a fallback plan should progress deviates from plans and estimates. The MasterNet project did not have any such fallbacks, controls or engage in prioritization and expectations management. It was the prediction of the theory that these factors combined would steer the project towards failure. In the case of MasterNet, the theory's prediction appeared to be consistent with the actual outcome of the project.

Case 3: MS Word for Windows

Background

This case study was published in (Kemerer, 1997). Microsoft introduced its first word processor for the PC called the PC Word in 1983. Having been disappointed with lukewarm reviews and mediocre sales, Bill Gates in 1984 initiated the development of a new state-of-the-art word processor for its Windows operating system codenamed Cashmere (later changed to Opus). Although its schedule slipped significantly from the originally projected ship date, Microsoft shipped Word for Windows with sales exceeding its own projections, and received significant critical acclaim press – it was Microsoft's first word processor to be rated higher than its competitor WordPerfect by InfoWorld.

Analysis Summary

Like the previously discussed cases, Microsoft's Opus project did a few things well, and overlooked a few as well. For example, the project did not spend a considerable amount of effort in prototyping, or identifying some of the critical product interdependencies in developing a word processor for their Windows operating system. On the other hand, Opus was extremely effective in adapting to changes, and in implementing necessary process and monitoring controls as the need emerged.

A significant difference in the case of this project from the others discussed above was the in the level of risk the project entailed. While true that success of Opus would make Microsoft a leader in office applications, and also complement sales of other Microsoft products, failure in delivering Opus would not have severely affected sales of Windows, Microsoft's flagship product. This observation is also consistent with the fact that while Opus kept missing deadlines, other than some negative media coverage, Microsoft was still flourishing in its business of other products.

Some other key risk reducing factor in the case of Opus was the experience Microsoft carried in building word processors for the Mac, availability to discretionary funds, high levels of commitment towards risk by senior management, fairly stable market conditions, and finally and most importantly the company's experience with managing risk in the development of technically challenging products.

Based on these observations, the application of theory predicted success contingent on these risk-reducing factors remaining stable, but at the same time also predicted adverse outcomes in terms of meeting deadlines – mainly attributed towards lack of efforts in prototyping or upfront planning. These predictions were consistent with the actual outcome of the project. Microsoft successfully delivered Windows for Word however in doing so it took five years versus the initial plan of on year.

Case 4: London Ambulance Service

Background

This case study was published in (Finkelstein, 1993). London Ambulance Service (LAS) was in the business of dispatching ambulances to individuals requiring urgent medical care. As such, when an emergency call is received by the LAS, ambulances are dispatched based on an understanding of the nature of the call and the availability of resources. In 1991, after LAS' management felt that an automated centralized ambulance service was essentially the way forward, it formed a small teams and assigned it the responsibility for generating a request for proposal (RFP), and thereby identify a contractor based on the meritocracy of each proposal submitted. Schedule and cost were the two most critical drivers for the team in identifying a suitable contractor.

After spending a few months developing an initial set of specifications for such an automated systems that formed the basis of LAS' RFP, the cheapest tender was accepted and a new system was developed and introduced the following year. The newly developed system however turned out to be a complete failure, resulted in a few casualties and was eventually rejected by LAS. Today, it is considered as a classic point of reference for "software disasters".

Analysis Summary

Many software systems fall into the category of "socio-technical systems" – a term loosely used to describe systems that permeate deep into the human ecosystem. Such systems affect human behavior, and often penetrate into a society that requires change at various levels. While today most software systems are perhaps socio-technical systems however, the term is usually used to classify systems when its impact on society is much greater than other systems. Many public welfare systems fall into this category, and special care must be given in designing and developing such systems.

For example, consider a system design scenario when there are two incidents requiring immediate medical attention however only one ambulance is available. Therefore projects engaged in building such systems carry enormous risks, and the theory recommends that a strong risk management plan be developed to help steer the project.

LAS' approach towards risk management left much to be desired. The CAD project ignored many success-critical stakeholders and their value propositions; failed to identify all of the key interdependencies across various dimensions; did not establish a win-win set of plans or establish feasibility and commitment with success-critical stakeholders; made value-agnostic and risk-agnostic decisions; and failed to orchestrate instruments of monitoring and control. The theory predicted

catastrophic failure, again consistent with the actual outcome as reported in the case study.

Case 5: CMU Surface Assessment Robot

Background

This case study was published in (Latimer, 2007). The study examines a project undertaken by CMU as a contractor to build a robot that can inspect the smoothness of a road surface while maintaining the same inspection quality as the manual method historically employed. While the project successfully delivered a robot that was designed right to the specifications, and if made operational would have also satisfied the estimated return on investment, it was however not transitioned into operational use.

Analysis Summary

One distinguishing element of the 4+1 theory from many other software engineering theories is the emphasis it places on organizational dependencies. CMU's project on constructing a surface assessment robot is an excellent case in point where an organizational dependency when left unattended jeopardized the entire project from achieving success.

This project had taken extreme care in identifying and involving different stakeholders in the development of a surface assessment robot. Requirements

analysis, prototyping, solid designs, and thorough implementation were some of the highlights of the project. Both technical and management decisions were made in a win-win environment, and the case per se had little evidence if any, to the contrary.

However, when VBSE theory was applied to this case, it predicted failure. Two primary reasons for this was 1) the acquiring organization was in the process of being acquired by another organization resulting in significant shifts in organizational control and 2) new stakeholders emerging from this acquiring organization were not identified as a success-critical stakeholder. This therefore led to key changes in stakeholder value propositions, and thereby destabilizing the win-win equilibrium of the project.

From the theory's point of view this particular case was unique because it helped illustrate the theory's completeness, and once again its ability to identify success from failure.

Insights

Analyzing the above-discusses case studies by way of applying the theory and process appeared to be extremely productive. Predictions made by the theory were consistent with the outcomes documented by the authors of the case studies. Some further insights gained in conducting this analysis seemed to be worth noting in light of future work on this topic. These will be discussed next.

Preferences, risk and maturity

Some of the analyzed case studies such as UniWord, MasterNet and London Ambulance omitted many of the theory recommended steps early into the project. Further analysis revealed that the impact of such omissions create a very strong ripple effect downstream, making the project almost unmanageable. While this observation appeals to intuitive reasoning, it also brings some additional situations that theory should address.

For example, a project that has suffered a trouble start in its inception will require much more risk management downstream than others. In such extreme situations, the theory may suggest a more risk-taking behavior assuming that the company is mature in handling high risk situations. However, certain organizations are more risk-averse than others – sometimes because they lack the adeptness or experience required to manage extreme risk or, simply as a matter of preference. The converse is true as well. Certain organization may prefer risk-taking behavior regardless of their maturity in handling risky situations while the theory may suggest a more risk-averse approach.

In the case of UniWord for example, after having discovered that the project is significantly off schedule and that the stakeholders' value function exhibits a step function (value heavily diminishes after a certain cut-off point), SoftWizards continued to expend critical resources in fixing the project – clearly

exhibiting a risk-taking behavior while not having the required experience and maturity in managing high risk.

In summary, this analysis recommends that future work should also consider complimenting decision theory to account for both, risk preferences as well as an organization's maturity to manage high risk.

Case 6: SMB with VBSE

In the previous cases, each case study was reviewed to show how the theory and the process proposed in this study could have identified and avoided the problems in each of the cases presented above. In this section, project SMB-0 as in ("SMB-0", 2005) is recreated with the same backdrop but using the theory and the process. It shows how SMB-0 could have been implemented using the theory and its derived process framework.

Sierra Mountainbikes Opportunities and Problems

Grant Golden began by convening his management and technology leaders, along with a couple of external consultants, to develop a constructive shared vision of Sierra Mountainbikes' primary opportunities and problems. The results determined a significant opportunity for growth, as Sierra's bicycles were considered top quality and competitively priced. The major problem area was in Sierra's old manual order processing system. Distributors, retailers, and customers were very frustrated with the high rates of late or wrong deliveries; poor

synchronization between order entry, confirmation, and fulfillment; and disorganized responses to problem situations. As sales volumes increased, the problems and overhead expenses continued to escalate.

In considering solution options, Grant and his Sierra team concluded that since their primary core competence was in bicycles rather than software, their best strategy would be to outsource the development of a new order processing system, but to do it in a way that gave the external developers a share in the system's success. As a result, to address these problems, Sierra entered into a strategic partnership with IPC for joint development of a new order processing and fulfillment system. IPC was a growing innovator in the development of supply chain management systems (an inventor with ideas looking for protagonist leaders with compatible goals and resources to apply their ideas).

Step 2: Identifying the Success-Critical Stakeholders (SCSs)

Step 2 in the VBSE process shown in Figure 1 involves identifying all of the success-critical stakeholders involved in achieving a project's goals. As seen in Figure 2, the Step 2a Benefits Chain jointly determined by Sierra and IPC, this includes not only the sales personnel, distributors, retailers, and customers involved in order processing, but also the suppliers involved in timely delivery of Sierra's bicycle components.

The Benefits Chain includes initiatives to integrate the new system with an upgrade of Sierra's supplier, financial, production, and human resource management information systems. The Sierra-IPC strategic partnership is organized around rewards reflecting both the system's benefits chain and business case, so that both parties share in the responsibilities and rewards of realizing the system's benefits. Thus, both parties share a motivation to understand and accommodate each other's value propositions or win conditions and to use value-based feedback control to manage the program of initiatives.

This illustrates the "only if" part of the Fundamental System Success Theorem. For the "if" part, if Grant had been a traditional cost-cutting, short-horizon executive, Sierra would have aggressively contracted for a lowest-bidder, fixed-price order processing system, and would have ended up with a buggy, unmaintainable stovepipe order processing system and many downstream order-fulfillment and supplier problems to plague its future. In terms of the framework in Figure 18, however, Sierra and eServices used the Benefits Chain form of Dependency Theory to identify additional SCSs (sales personnel, distributors, retailers, customers, suppliers) who also need to be brought into the SCS WinWin equilibrium state.

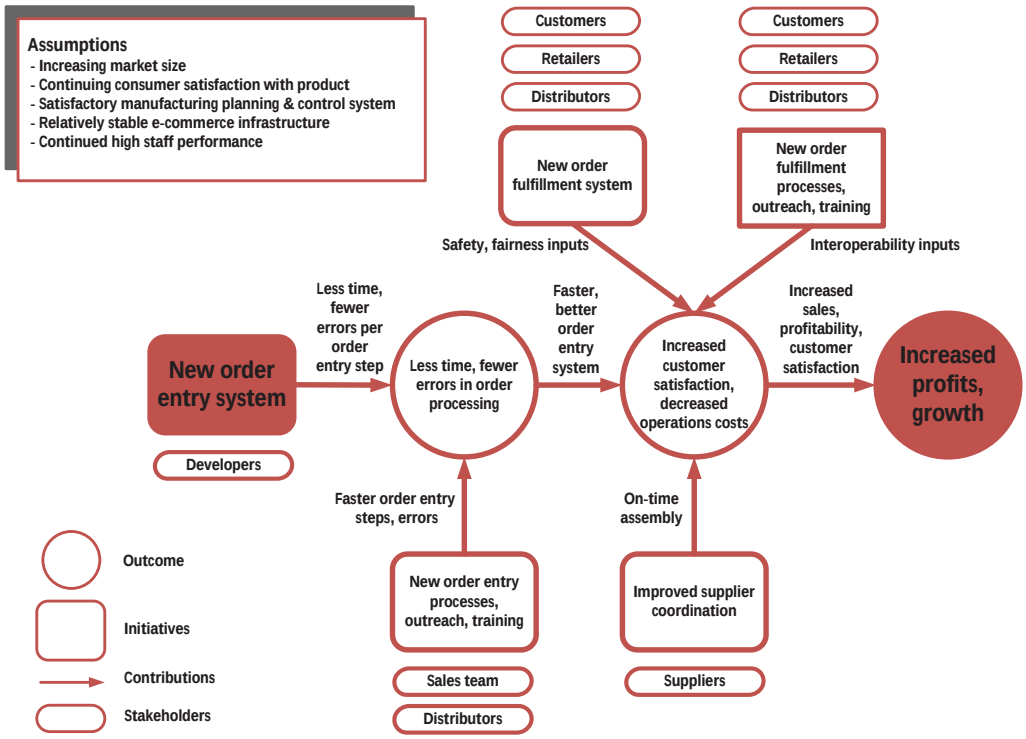


Figure 18. Benefits Chain for Sierra Supply Chain Management

Step 3: Understanding SCS Value Propositions

Step 3 primarily involved utility theory. But it also involved Theory W in reconciling SCS win conditions with achievable solutions (Step 3a), and various forms of dependency theory in conducting cost/schedule/performance solution tradeoff and sensitivity analyses (Step 3b).

For example, the suppliers and distributors may identify some complex exception reporting, trend analysis, and customer relations management features they would like to have in the system's Initial Operational Capability (IOC) in early 2005. However, the use of forms of dependency theory such as software cost and

schedule estimation models may show that there is too much proposed IOC software to try to develop by the IOC date.

In such a case, Sierra and IPC will have to revisit the SCSs' utility functions in Step 4a (expectations management) by showing them the cost and schedule model credentials and results, and asking them to recalibrate their utility functions, prioritize their desired features, and participate in further solution exploration (a go-back to Step 3a) to achieve a win-win consensus on the top-priority subset of features to include in the IOC.

It may be in some cases that the SCSs' IOC needs are irreconcilable with the IOC schedule. If so, the SCSs may need to live with a later IOC, or to declare that a SCS win-win state is unachievable and to abort the project. Again, it is better to do this earlier rather than later.

Step 4: Managing Expectations; SCSs Negotiate a WinWin Decision

Actually, the previous paragraph anticipates the content of Step 3, in which the SCSs negotiate a win-win decision to commit themselves to go forward. Once the SCSs have identified and calibrated their Win Conditions in Step 3 and 4a, the process of identifying conflicts or Issues among Win Conditions; inventing and exploring Options to resolve Issues; and converging on Agreements to adopt Win Conditions or Options proceeds as described in the WinWin Negotiation Model above.

In a situation such as the Sierra supply chain project, the number of SCSs and the variety of their win conditions (cost, schedule, personnel, functionality, performance, usability, interoperability, etc.) means that multi-attribute decision theory will be involved as well as negotiation theory. Susan will also be concerned with investment theory or business case analysis to assure her stakeholders that the supply chain initiative will generate a strong return on investment. As many of the decisions will involve uncertainties (market trends, COTS product compatibilities, user interface choices), forms of statistical decision theory such as buying information to reduce risk will be involved as well.

User interface prototypes are actually ways of buying information to reduce the risk of misunderstanding SCS utility functions, as indicated in Figure 17 by the arrow between decision theory and utility theory. The other components of Step 4b in Figure 1 involve other aspects of dependency theory, such as performance analysis, business case analysis, or critical-path schedule analysis. As also shown in Figure 17, these analyses will often proceed at increasing levels of detail in supporting steps 3a, 4a, and 6a as the project proceeds into detailed design, development, integration, and test.

Table 6 summarizes the business case analysis for the Sierra project. Dollar values are all in millions of 2004 dollars ($M) for simplicity. The analysis compares the expected sales and profits for the current system (columns 4, 5) and the new system (columns 7, 8) between 2004 and 2008, the cumulative increase in profits, investment cost, and resulting return on investment (columns 11-13), and expected improvements in other dimensions such as late delivery and customer satisfaction (columns 14-17). The bottom line is a strong 2.97 ROI, plus good expected outcomes in the customer satisfaction dimensions.

The negotiations converge on a number of win-win agreements, such as involving the suppliers and distributors in reviews, prototype exercising, and beta-testing; having Sierra provide eServices with two of their staff members to work on the software development team; and agreeing on compatible data definitions for product and financial interchange. At one point in the negotiation, an unfortunate go-back is necessary when an Agreement on a product definition standard is reversed by the management of one of the distributors, who disclose that they are now committed to an emerging international standard. After some renegotiation, the other SCSs agree to this at some additional cost.

Table 6. Expected Benefits and Business Case

		Date	12/31/03	12/31/04	12/31/05	12/31/06	12/31/07	12/31/08
		Market Size ($M)	360	400	440	480	520	560
CURRENT SYSTEM		Market Share %	20	20	20	20	20	20
		Sales	72	80	88	96	104	112
		Profits	7	8	9	10	11	12
NEW SYSTEM	Financials	Market Share %	20	20	22	25	28	30
		Sales	72	80	97	120	146	168
		Profits	7	8	10	13	16	19
		Cost Savings	0	0	2.2	3.2	4	4.4
		Change in Profits	0	0	3.2	6.2	9	11.4
		Cum. Change in Profits	0	0	3.2	9.4	18.4	29.8
		Cum. Cost	0	4	6	6.5	7	7.5
		ROI	0	-1	-0.47	0.45	1.63	2.97
	Customers	Late Delivery %	12.4	11.4	7	4	3	2.5
		Customer Satisfaction (0-5)	1.7	3	4	4.3	4.5	4.6
		Ease of Use (0-5)	1.8	3	4	4.3	4.5	4.6
		In-Transit Visibility (0-5)	1	2.5	3.5	4	4.3	4.6

Steps 5 and 6: Planning, Executing, Monitoring, Adapting, and Controlling

As with the dependency analyses, project planning, executing, monitoring, adapting, and controlling proceed incrementally in increasing amounts of detail, generally following a risk-driven spiral process. Questions such as "how much is enough planning, specifying, prototyping, COTS evaluation, business case analysis, architecting, documenting, verifying, validating etc.?" are best resolved by balancing the risk exposures of doing too little or too much. As Risk Exposure = Probability (Loss) * Value (Loss) is a value-based concept, risk balancing is integral to the theory.

Value-based planning and control differs most significantly from traditional project planning and control in its emphasis on monitoring progress toward value realization rather than towards project completion. Particularly in an era of increasing rates of change in market, technology, organizational, and environmental conditions, there is an increasing probability that managing to a fixed initial set of plans and specifications will produce systems that are out of step and non-competitive with projects managing adaptively toward evolving value realization.

Traditional control mechanisms such as the earned value management, assigns "value" to the completion of project tasks and helps track progress with respect to planned budgets and schedules, but has no way of telling whether

completing these tasks will add to or subtract from the business value or mission value of the enterprise. Example failure modes from this approach are systems that had to be 95% redeveloped on delivery because they failed to track evolving requirements and startup companies that fail to track closure of market windows.

If an organization has used steps 1-4 to identify SCSs, determine their value propositions, and develop business cases, it has developed the framework to monitor expected value realization, adjust plans, and control progress toward real SCS value achievement. Table 7 shows how this could be done for the Sierra project, based on the initial budgets, schedules, and business case in

Table 6. The planned achievables are above the line in each cell of Table 7; the actuals are below. Value-based monitoring and control for Sierra requires additional effort in terms of technology watch and market watch, but these help Sierra to discover early that their in-transit-visibility (ITV) COTS vendor was changing direction away from Sierra's needs.

This enabled Sierra to adapt by producing a timely fallback plan, and to proactively identify and approach other likely ITV COTS vendors. The results, as shown in the ITV column and explained in the Risks/Opportunities column of Figure 4, was an initial dip in achieved ITV rating relative to plans, but a recovery to close to the originally planned value. The Risks/Opportunities column also shows a "new hardware competitor" opportunity found by market watch activities that results in a $200K hardware cost savings that mostly compensated for the added software costs of the ITV fallback. The use of prioritized requirements to drive value-based Pareto- and risk-based inspection and testing is another source of software cost savings.

Table 7. Value-Based Expected/Actual Outcome Tracking

		Schedule	Cost ($K)	Op. Cost Savings	Market Share %	Annual Sales ($M)	Annual Profits ($M)	Cum. Profits	ROI	Late Delivery %	Customer Satisfaction	ITV	Ease of Use	Risks/Opportunities
MILESTONES	Life Cycle Architecture	3/31/04	400		20	72	7			12	1.7	1	1.8	1
		3/31/04	427		20	72	7			12	1.7	1	1.8	
	Core Cap. Demo	7/31/04	1050											2
		7/20/04	1096								**2.4**	**1.0**	**2.7**	
	SW Init. Op. Cap. (IOC)	9/30/04	1400											
		9/30/04	1532								**2.7**	**1.4**	**2.8**	
	Hardware IOC	9/30/04	3500											3
		10/11/04	3432											
	Deployed IOC	12/31/04	4000		20	80	8	0	-1	11	3	2.5	3	4
		12/20/04	4041		22	88	8.6	0.6	-1	11	2.8	1.6	3.2	
	Responsive IOC	3/31/05	4500	300						9	3.5	3	3.5	
		3/30/05	4604	324						7.4	3.3	1.6	3.8	
	Full Op. Cap. CCD	7/31/05	5200	1000							**3.5**	**2.5**	**3.8**	5
		7/28/05	5328	946										
	Full Op. Cap. Beta	9/30/05	5600	1700							**3.8**	**3.1**	**4.1**	
		9/30/05	5689	1851										
	Full Op. Cap. Deployed	12/31/05	6000	2200	22	106	12	3.2	-0	7	4	3.5	4	
		12/20/05	5977	2483	24	115	14	5.1	-0	4.8	4.1	3.3	4.2	
	Release 2.1	6/30/06	6250											

(1) Increased COTS ITV risk, fallback identified.
(2) Using COTS ITV fallback; new HW competitor; renegotiating HW.
(3) $200K savings from renegotiated HW.
(4) New COTS ITV source identified, being prototyped.
(5) New COTS ITV source initially integrated.
(**bold**) Interim ratings based on trial use

The bottom-line results are a good example of multi-attribute quantitative/qualitative balanced-scorecard methods of value-based monitoring,

adaptation, and control. They are also a good example of use of the necessary conditions for value-based control based on control theory. A traditional value-neutral "earned value" management system would fail on the criteria of business-value observability, predictability, and controllability, because its plans, measurements, and controls deal only with internal-project progress and not with external business-value observables and controllables. They also show the value of adaptive control in changing plans to address new risks and opportunities, along with the associated go-backs to revisit previous analyses and revise previous plans in Steps 6a, 6b, and 6c.

Chapter 6: Conclusions and Recommendations

Review of Problem

In chapter 1, this study made a case for using value-based approaches in the development of software systems. Two problems with existing approaches were identified – value-disconnect and context disconnect. Value disconnect is the phenomenon when decisions about software systems are made without incorporating success-critical stakeholder values. A software system is a means to an end, not an end per se. Stakeholder values are the ends that the software system must deliver. If stakeholder values are not incorporated in such decisions, then it essentially becomes a matter of chance that random decisions will lead to realizing the stakeholder values.

A context disconnect refers to the phenomenon when a software system's context such as the organization and its environment are not taken into consideration. Chapters 1 and 3 showed how a software system is often dependent on many organizational characteristics, such as the organization's strategy, management style, group climate, structure, size, and environment. These dependencies could either serve as constraints on the way software systems are developed, or provide guidance for realizing greater benefits. In either

case, a software system's context must be included in decision making and control.

Review of Purpose

The purpose of this research was to see if a value-based theory for software engineering be developed that provides (a) criteria that distinguish projects that will fail from projects that will succeed and (b) a process that applies the criteria to enable projects to succeed and satisfactorily addresses the criteria for a good theory.

For validation, a combination of logical analysis, integration of well-established theories, elaboration of the less well-established dependency theory, and use of available quantitative results in value-based software engineering was performed in Chapters 3 and 4. In Chapter 5, these results were complemented by the analysis of six diverse case studies.

Review of Results

In Chapter 3, semiformal proofs of the two fundamental theorems of value-based software engineering were developed. The Fundamental System Success Theorem establishes success-critical stakeholder win-win conditions for creating a successful system.

The System Success Realization Theorem establishes conditions for making winners of the success-critical stakeholders:

(a) Identifying all of the SCSs.

(b) Understanding how the SCSs want to win.

(c) Having the SCSs negotiate a win-win set of product and process plans.

(d) Controlling progress toward SCS win-win realization, including adaptation to change.

These were mapped onto the major theories supporting the four conditions: dependency theory, utility theory, decision theory, and control theory.

In Chapter 4, a 4+1 architecture was developed for the main theory and the four supporting theories, and a 6-step process was derived for projects to use in applying the theory.

In chapter 5, six case studies were reviewed using a set of questions derived independently for each of the 4+1 theories (see Appendix A for the list of questions). In all the six cases, the theory was able to diagnose the project in terms of success and failure. For example, in the UniWord and MasterNet case studies, it was observed that the project not only failed to identify all of its success-critical stakeholders, it also failed to identify the organizational dependencies which in turn allowed the project to make decisions in isolation from the organization and its needs. As such, the prediction of the theory

corroborated with the outcome of the projects, both the projects were unsuccessful.

Relations to Criteria for a Good Theory

The study reviewed a number of sources of criteria for a good theory, and converged on the following composite list for evaluating the theory: sufficiency, necessity, utility, generality, practicality, preciseness, and falsifiability. A summary of the resulting evaluations is provided next.

Sufficiency

According to Swartz (1997):

> "A condition A is said to be sufficient for a condition B, if (and only if) the truth (/existence /occurrence) [as the case may be] of A guarantees (or brings about) the truth (/existence /occurrence) of B."

That is, in the context of this study, to show that the five theoretical constructs were sufficient for the theory. In chapter 5, the theory was applied to multiple case studies. In each case, the theory's prediction corroborated with the actual outcome of the case. Further, an alternative version of SMB-Zero was reconstructed as SMB-VBSE to show that the theory was sufficient in guiding the project across all situations that had originally developed in SMB-Zero.

An additional note on the equifinality characteristic of the sufficiency condition must also be made. Sufficiency is established when A is enough for B (A

is sufficient for B if and only if A brings about B) however, there may be another way, say C, that could also bring about B. Since the theory brought about the desired outcomes in the case studies, it is sufficient in their contexts.

However, logical analysis of the sufficiency proof for the System Success Realization Theorem indicates that conditions (a) through (d) may not always be realizable. In particular, for condition (c), fundamental differences may exist among the success-critical stakeholders, making it impossible to negotiate a win-win outcome. Also, for condition (d), an unpredictable show-stopper may impact the project, such as a loss of budget or a radical change in executive personnel and priorities. In such cases, the theory is still valuable in detecting such situations early and minimizing adverse impacts of terminating the project.

Necessity

According to Swartz (1997):

> "A condition A is said to be necessary for a condition B, if (and only if) the falsity (/nonexistence /non-occurrence) [as the case may be] of A guarantees (or brings about) the falsity (/nonexistence /non-occurrence) of B."

That is, in the context of this study, to show that all the five theoretical constructs are necessary for the theory. In Chapter 2, a choice rationale was provided which offered two explanations to show that each of the five theoretical constructs were necessary for a theory of developing software systems. In the first explanation, Theory W was used as a maximized goal of the theory, and Theory

W's predicates led to the choice of the four remaining constructs. In the second explanation, an argument was made to show how normative decision making per se leads to the choice of decision, utility and dependency theories, and that since in most cases the environment in which a software system is developed and deployed is not static, controlling development is imperative. As such including control theory was valid.

However, logical analysis also identified situations in which a poorly-managed project could violate all four of the success realization conditions, and still emerge successfully. For example, the project could have a satisfactory COTS product become available just in time to be installed as the system solution.

Utility

Utility involves addressing the project's critical success factors in cost-effective ways. Several techniques discussed in the process case study provide examples of cost-effective approaches. For example, the Results Chain method in Step 2 helped identify the missing success-critical initiatives and stakeholders in an efficient way. For another example, the risk-driven inspection and test approaches in Step 5 avoided wasting inspection and test time on trivial-value aspects of the system. However, further progress could be made in providing such techniques, as discussed under topics for further research.

Generality

Generality is that it covers procedural, technical, economic, and human concerns; and covers small and large systems. The 6-step process and its ability to accommodate parallel activities and go-backs were sufficient to cover the Sierra project's procedural needs. Technical and economic concerns are addressed in the use of dependency theory for cost, schedule, performance, and business case analyses in Steps 3a, 4a, and 6b. Human concerns are the essence of Theory W and utility theory, and of the SCS negotiations in Step 4. The six different case studies had a variety of different organizational and project characteristics, and the steps of the theory worked well for all of them. Again, further research would help evaluate the generality of the theory and process.

Practicality

Practicality is that it supports practical needs for prediction, diagnosis, solution synthesis, good-practice generation, and explanation. The theory draws on a wide-variety of dependency models (e.g. cost, schedule, performance, quality) to predict outcomes. In a stable, well-understood environment, managing to the predictions usually produces a self-fulfilling prophecy. In less stable and less familiar situations such as the Sierra case study, dependency theory was able to diagnose risks such as missing stakeholders in Step 2, Theory W was able to support synthesis of SCS win-win solutions in Step 4, and adaptive control theory

was able to generate good value-achievement monitoring practices to support in-process diagnosis and re-synthesis in Steps 5-6. The control theory necessary conditions of observability and controllability were able to explain why traditional earned value systems would not have addressed and resolved these value-domain problems. Here again, further research and usage would likely provide additional practical capabilities and evidence of practicality.

Preciseness

Preciseness is that it provides situation-specific and accurate guidance. The theory is no more (and no less) accurate than its constituent theories in predicting outcomes of unprecedented situations, but it is able to provide situation-specific guidance, as shown in its application to the Sierra supply-chain project. Also, several examples were provided in the SMB-VBSE case study to show how the theory would have generated different guidance in different situations, such as with the distributor management's reversal of a win-win agreement on a product definition standard in Step 4, and with the ITV COTS vendor's change of direction in Steps 5 and 6.

Falsifiability

Falsifiability is the ability to be empirically refuted. The SMB-VBSE case study identified a particular situation in which application of the theory could not produce a win-win solution, leading to a timely decision to cancel the project. This

involved incompatible and non-negotiable SCS win conditions about Initial Operational Capability content and schedule in Step 4. A similar outcome could have resulted from the distributor management change of direction in Step 5.

Additionally, there are several other situations where the theory will not work. Some such situations include when:

- People disguise their true win conditions
- People like to win by making others losers
- There can only be one winner

However, many apparent only-one-winner or zero-sum-game situations can be turned into win-win situations by expanding the option space. A good example is provided in *Getting to Yes* (Fisher-Ury, 1981), in which a boundary-line location stalemate on ownership of the Sinai Desert between Egypt and Israel was resolved by creating a new option: the land was given back to Egypt, satisfying its territorial win condition, but it was turned into a demilitarized zone, satisfying Israel's security win condition.

Implications

As discussed above, the theory has proved in logical analysis to be necessary and sufficient, except for the special cases discussed above. It also proved sufficient in application to the six case studies. Overall, the theory provides

a robust framework for decision makers to reason about their project decisions, and to assess potential value-domain risks in their approach.

Additionally, the 6-step process framework provides principled process guidance for software engineers to implement value-based approaches in the development of software systems.

Future Work

To successfully hand over the baton to any potential candidate interested in pursuing this research further, it is imperative that the experiential knowledge gained is also made available to such a candidate. To this end, this section discusses some of the challenges associated with continuing this study, explains why this study is an attractive proposition, and finally offers a few low hanging opportunities to any such interested candidate.

Challenges

There are many significant challenges that lay ahead for any successor of this research. This is because (1) process and management research in engineering and elsewhere typically takes over a decade to become validated on large end-to-end projects; (2) an interdisciplinary theory as proposed in this study requires an enormous amount of time and other resources that are often severely limited in dissertation-type research projects; (3) the "broadness" of this theory – involving the intricate details of organizational characteristics, external environment,

people, process and product, is its strength but also its weakness, at least in terms of practicality of validation. It is very hard to find good data even within a project, let alone the organization and institution levels; (4) the "interdisciplinary" nature of this theory has some serious consequences for continuing it as dissertation-type research. For example, when done within the confines of "engineering", many traditional-engineering dissertation committees will find such theories to be beyond the scope of engineering. Additionally, inconsistencies in terminology and ideology across various disciplines not only make convergence hard, but also extremely time consuming. Bearing these challenges in mind, perseverance and an iterative approach is highly recommended.

A Case for Future Work

In spite of all the challenges, a strong case can still be made in continuing this research. First, this research is novel. Very little (some of the exceptions include the chapters in Biffl et al. (2005); Boehm and Turner (2004); Rifkin (2006); Huang and Boehm (2006)) has been said about the role of stakeholder values and organization context in developing software systems. In particular, application of contingency theory (as an organization theory), to software systems engineering has been extremely limited (except for Rifkin (2006), but even his application was very limited).

Two, there are enormous opportunities for making strong contributions in engineering, and perhaps in other fields as well. Decisions about software systems today are largely made in a value-restricted flatland, where the key differentiators between choices are either "institutional" factors (right vs. wrong, legitimacy etc.), or "traditional" factors (limited to cost, schedule, and defect rate). These factors can be sufficient in some situations while completely ineffective in others. Value-based approaches provide better reasoning tools for decision making, and thus expectedly yield better outcomes.

Low Hanging Opportunities

Many avenues exist in strengthening this research – from further explicating each of the theoretical constructs to conducting experiments for quantitative analysis. However, some of these are more attractive than the rest. This study recommends that future work should focus on at least the following two dimensions – (1) enhancing robustness of the theory and (2) focusing on its applicability (gearing for practice) to at least some of the success critical decisions that are made in the development of software systems.

Theory Robustness

Robustness of the theory refers to the resilience of the theory in unprecedented situations. One approach for enhancing robustness of the theory is to examine another sample set of case studies and determining under what

conditions it would have failed. Another approach would be to capture experiential knowledge through forms such as peer reviews and wideband delphis. This study uses contingency theory as a form of dependency theory. However, as discussed in Chapter 2, there are at least two other well developed theories that can serve as a form of dependency theory – institutional theory and resource dependency theory. Application of either or both may have very attractive returns in understanding the limitations of engineering methods for developing software systems. For example, contingency theory does not explain compliance, legitimacy, or mimetic behaviors of organizations as explained by institutional theory.

Within the realm of contingency theory too, there are many potential opportunities for increasing robustness. For example, Burton and Obel (2004) identify at least 20 different organizational contingencies (dependencies) that could potentially affect how software systems are developed. This study's scope involved discussion of only few of them were discussed, and with limited empirical support. Each such dependency (for example highly centralized structure with risk-averse management style) is a research topic per se, and requires empirical evidence.

Additionally, the concept of power struggle is not addressed in this study since there has been little said about the role of power by contingency theorists. It would be unwise to ignore it, albeit there is an implicit notion of power in Burton

and Obel's (2004) management style and structural factors. Even if alternative approaches to contingency theory (such as institutional theory) are not examined, it may be useful to know the extent to which power, legitimacy etc., threaten the validity of this study. Finally, and most importantly, all assumptions made in this study should be revisited as more media of validation become available.

Gearing for Practice

The future of the VBSE theory depends on the extent to which it captures attention from the practitioner community. To attract practitioners, future studies must also consider how best to package this study (theory and process framework) in forms that will make direct contribution to some of the problems practitioners face in decision making.

For example, Huang and Boehm's (2006) value-based software quality assurance (VBSQA) model shows how stakeholder values can be incorporated into the various investment decisions on testing and system quality. Similarly, models for making investment decisions with respect to requirements and negotiation, architecture and design, construction, will significantly add value to this study, and make it attractive to practitioners.

Some other attractive specialty areas that could benefit from further value-based extensions are value-based project monitoring and control, risk management, product line reuse, software maintenance, corporate knowledge management, and intellectual property management. Some initial results in these

directions can be found in the chapters of Biffl et al. (2005). Further practical research would involve elaboration of key value-based practices, such as rapid determination of stakeholders' value propositions and priorities; techniques for creating options for mutual gain (some are provided in Fisher and Ury (1981) and Boehm and Ross (1989)); and elaboration and specialization of the key value-based practices to provide domain-specific guidance for particular application domains. Such domain-specific research would frequently provide additional insights for the general theory and process.

References

Ahern, D. M., Clouse, A., Turner, R. CMMI Distilled: A Practical Introduction to Integrated Process Improvement, Addison-Wesley, 2003.

Alexander, C., The Timeless Way of Building, Oxford University Press, 1979.

Al-Said, M., "Model Clashes", Ph.D. Dissertation, University of Southern California, 2003

Argyris, C., Organizational Learning, Addison-Wesley, 1978.

Austin, R., Measuring and Managing Performance in Organizations, Dorset House 1996.

Babcock, C., "New Jersey Motorists in Software Jam," ComputerWorld, vol. September, pp. 1-6, 1985.

Baldwin, C., Clark, K., Magretta, J., Dyer, J., Fisher, M., and Fites, D., Managing the Value Chain, Harvard Business School Press, 2000.

Bell, D., Raiffa, H., and Tversky, A. (eds.), Decision Making: Descriptive, Normative, and Prescriptive Interactions, Cambridge University Press, 1988.

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

Blackwell, D., Girshick, M., Theory of Games and Statistical Decisions, Wiley, 1954.

Blanchard, B. and Fabrycky, W., Systems Engineering and Analysis, Prentice Hall, 1988.

Boehm B., Jain, A., "A Value-Based Theory of Systems Engineering," in Proceedings of INCOSE Symposium Orlando, Florida, 2006.

Boehm B., Jain, A., "A Value-Based Theory of Systems Engineering: Identifying and Explaining Dependencies", Proceedings, INCOSE, 2007.

Boehm B., Jain, A., “An Initial Theory of Value-Based Software Engineering”, in Value-Based Software Engineering, Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Gruenbacher, P. (eds.), Springer Verlag, 2005, pp 15-37.

Boehm B., Turner, R., Balancing Agility and Discipline: A Guide for the Perplexed, Addison Wesley, 2004.

Boehm, B. et al., Characteristics of Software Quality, TRW Report to NBS, 1973 (also North Holland, 1978).

Boehm, B., "Boehm's Top 10 Risk List" (http://csse.usc.edu/BoehmsTop10), accessed 03/29/2007.

Boehm, B., “A Spiral Model for Software Development and Enhancement”, IEEE Computer, pp. 61-72, 1988.

Boehm, B., “Software and Its Impact: A Quantitative Assessment”, Datamation, pp 48-59, 1973.

Boehm, B., and Hansen, W. "The Spiral Model as a Tool for Evolutionary Acquisition", CrossTalk, May 2001.

Boehm, B., Bose, P., “A Collaborative Spiral Software Process Model Based on Theory W”, Proceedings, ICSP 3, IEEE, October 1994.

Boehm, B., Huang, L., “Value-Based Software Engineering: A Case Study”, IEEE Computer, March 2003, pp. 21-29.

Boehm, B., In, H., "Identifying Quality-Requirement Conflicts", IEEE Software, Vol. 13, No. 2, pp. 25-35, March 1996.

Boehm, B., Port, D., Al-Said, M., "Avoiding the Software Model-Clash Spiderweb," Computer, vol. 33, no. 11, pp. 120-122, 2000.

Boehm, B., Ross, R., “Theory-W Software Project Management: Principles and Examples”, IEEE Transactions in Software Engineering, pp 902-916, July 1989.

Boehm, B., Software Engineering Economics, Prentice Hall, 1981.

Boehm, B., Tutorial: Software Risk Management, IEEE Computer Society Press, 1989.

Booher, H. (ed.), Handbook of Human Systems Integration, Wiley, 2003.

Borchers, G., "The Software Engineering Impacts of Cultural Factors on Multi-cultural Software Development Teams", Proceedings, ICSE, pp. 540 -545, 2003.

Brogan, W., Modern Control Theory, Prentice Hall, 1974.

Bullock, J., "Calculating the Value of Testing", Software Testing and Quality Engineering, Volume 2, Issue 3, pp. 56-62, 2000.

Burns, R., To a Mouse, November 1785.

Burns, T., Stalker, G. M., The Management of Innovation, Tavistok, 1961.

Burrell, G., Morgan, G., Sociological Paradigms and Organizational Analysis, Gower Publishing Company Limited, 1979.

Burton R.M. & Obel B., Strategic Organizational Diagnostics and Design: The Dynamics of Fit. Kluwer Academic Publishers, 2004.

Carr, D., "Sweet Victory", Baseline, December 2002.

Chandler, A. D., Strategy and Structure: Chapters in the History of the Industrial Enterprises, MIT Press, 1962.

Checkland, P., Systems Thinking, Systems Practice, Wiley, 1981.

Chung, L. et al., Non-Functional Requirements in Software Engineering, Kluwer, 1999.

Clements, P., et al., Documenting Software Architectures: Views and Beyond, Addison Wesley, 2003

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

Cohen, M. D., March, J. G., and Olsen, J. P., "A Garbage Can Model of Organizational Choice", Administrative Science Quarterly, vol. 17(1), pp. 1-25, 1972.

Crosby, P., Quality is Free, New York: McGraw-Hill, 1979.

Daft, R., Organization Theory and Design, South-Western College Publications, 8th ed., 2003.

Danto A. and S. Morgenbesser S. (eds.), Philosophy of Science, Meridian Books, 1960.

Dantzig, G., Linear Programming and Extensions, Princeton U. Press, 1963.

Debreu, G., Theory of Value, Wiley, 1959.

DeMarco, T., Controlling Software Projects: Management, Measurement, and Estimates, Yourdon Press, 1986.

Deming, W. E., Out of the Crisis, MIT Press, 1986.

DiMaggio, P. J., "Interest and Agency in Institutional Theory", in L. G. Zucker, ed., Institutional Patterns and Organizations, Ballinger, 1988.

DiMaggio, P. J., and Powell, W. W., "The Iron Cage Revisited: Institutional Isomorphism and Collective Rationality in Organizational Fields", American Sociological Review, vol. 48 (2), pp. 147-60, 1983.

DiMaggio, P. J., and Powell, W. W., The New Institutionalism in Organizational Analysis, University of Chicago, 1991.

Donaldson, L., The Contingency Theory of Organizations, Sage, 2001.

Dupuit, J., "On the Measurement of the Utility of Public Works", Translated by R. H. Barback, International Economic Papers 2:83-110, 1844 (1952).

Favaro, J., "When the Pursuit of Quality Destroys Value", IEEE Software, May 1996.

Finkelstein, A. and Dowell, J. "A Comedy of Errors: The London Ambulance Service Case Study", Proceedings, 8th International Workshop on Software Specification and Design, pp. 2, 1996.

Fishburn, P. C., The Foundations of Expected Utility, Dordrecht, 1982.

Fisher, R., Ury, W., Getting To Yes: Negotiating Agreement Without Giving In, Houghton Mifflin, 1981.

Flowers, S., Software Failure: Management Failure: Amazing Stories and Cautionary Tales, John Wiley and Sons, 1996.

Ford, L., Fulkerson, D. R., Flows in Networks, Princeton University Press, 1962.

Forrester, J.W., Industrial Dynamics, Productivity Press, 1961.

Galbraith, J., Designing Complex Organizations, Addison Wesley, 1973.

Galbraith, J., Organization Design, Addison Wesley, 1977.

Gioia, D. A., and Pitre, E., "Multi-paradigm Perspectives on Theory Building", Academy of Management Review, vol. 15, pp. 584-602, 1990.

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

Glass, R., Facts and Fallacies of Software Engineering, Addison Wesley, 2003

Goodrick, E., and Salancik G. R., "Organizational Discretion in Responding to Institutional Practices: Hospitals and Cesarean Births" in Administrative Science Quarterly, vol. 41(1), pp. 1-28, 1996.

Hempel, C. G. , and Oppenheim, P., "Problems of the Concept of General Law", in Danto, A. and Mogenbesser, S. (eds.), Philosophy of Science, Meridian Books, 1960.

Highsmith, J., Adaptive Software Development, Dorset House, 2000.

Huang, L. and Boehm, B., "How Much Software Quality Investment Is Enough: A Value-Based Approach", Boehm IEEE Software, vol. 23, no. 5, September/October, pp. 88-95, 2006.

Humphrey, W., Managing the Software Process, Addison Wesley, 1989.

J. Harsanyi, "A General Theory of Rational Behaviour in Game Situations", Econometrica, vol. 34, pp. 613-34, 1966.

Jones, C. , Software Development: A Rigorous Approach, Prentice Hall, 1980.

Jones, C., Assessment and Control of Software Risks, Yourdon Press Computing Series, New Jersey, 1994.

Juran, J. M., Gryna, F. M., Juran's Quality Control Handbook, Mcgraw-Hill, 1951.

Juristo, N., Moreno, A., and Acuna, S., A Software Process Model Handbook for Incorporating People's Capabilities, Kluwer, 2005.

Kaplan, R., Norton, D., The Balanced Scorecard: Translating Strategy into Action, Harvard Business School Press, 1996.

Kemerer, C. F., Software Project Management: Readings and Cases, McGraw-Hill, 1996.

Keeney, R. L., "Building Models of Values", European Journal of Operational Research, vol. 37, pp. 149-157, 1988.

Keeney, R. L., Raiffa, H., Decisions with Multiple Objectives: Preferences and Value Tradeoffs, Cambridge University Press, 1976.

Kenneth, A., "Behaviour under Uncertainty and its Implications for Policy", in Decision Making: Descriptive, Normative, and Prescriptive Interactions, Bell, D., Raiffa, H., and Tversky, A. (eds.), pp. 167-192, Cambridge University Press, 1988.

Larman, C., Agile and Iterative Development: A Manager's Guide, Addison Wesley, 2004.

Latimer, D., "Acquisition of Robotic System Capabilities", Ph.D. Dissertation Proposal, University of Southern California, 2007.

Lee, M. J., "Foundations of the WinWin Requirements Negotiation System", Ph.D. Dissertation, University of Southern California, 1996.

Leveson, N. G. & Turner, C. S., "An Investigation of the Therac-25 Accidents", Computer, vol. 26, no. 7, pp. 18-41., 1993.

LiGuo, H., Boehm, B., "How Much Software Quality Investment Is Enough: A Value-Based Approach", IEEE Software, vol. 23, no. 5, September/October, 2006.

Luce, R. D., Raiffa, H., Games and Decisions, John Wiley, 1957.

Luehrman, T. A., "Investment Opportunities as Real Options: Getting Started on the Numbers" Harvard Business Review, July/August, pp 51-67, 1998.

Mahoney, J. T., and Sanchez, R., “Building New Management Theory by Integrating Processes and Products of Thought.” Journal of Management Inquiry, vol. 13 (1), pp. 34-47, 2004.

Mahoney, M.S., "Finding a History for Software Engineering", IEEE Annals of the History of Computing, vol. 26-1, pp. 8-19, 2004.

March, J., and Simon, H., Organizations, Wiley, 1958.

March, J.G., Heath, C., A Primer On Decision Making: How Decisions Happen, Free Press, 1994.

Marschack, J., Radner, R. The Economic Theory of Teams, Yale University Press, 1972.

Maslow, A. H., “A Theory of Human Motivation”, Psychological Review, 50, pp. 370-396, 1943.

Maslow, A., Motivation and Personality, Harper, 1954.

McConnell, S., Rapid Development, Microsoft Press, 1996.

Meyer, J. W., Scott, W R., and Strang, D., "Centralization, Fragmentation, and School District Complexity." Administrative Science Quarterly, vol. 32(2), pp. 186-201, 1987.

Meyer, J., and Rowan, B., “Institutionalized Organizations: Formal Structures as Myth and Ceremony”, American Journal of Sociology, vol. 83, pp. 340-363, 1977.

Miller, R. W. and Collins, C. T., “Acceptance Testing”, Proceedings, XP Universe, 2001.

Mintzberg, H., Managers not MBAs: A Hard Look at the Soft Practice of Managing and Management Development, Berrett-Koehler Publishers, 2004.

Mintzberg, H., Raisinghani, D., and Théorêt, A. “The Structure of “Instructed” Decision Processes”, Administrative Science Quarterly, vol. 21 pp. 246-275, 1976.

Newnan, D., Lavelle, J., and Eschenbach, T., Engineering Economic Analysis, Oxford University Press, 9th ed., 2004.

Parsons, T., Social Systems and the Evolution of Action Theory, The Free Press, 1977.

Paulk, M. et. al., Interpreting CMM for Small/Prototyping Projects, SEI, April 1994.

Perrow, C., "The Short and Glorious History of Organizational Theory," Organizational Dynamics, pp. 2-15, 1973.

Perrow, C., Complex Organizations: A Critical Essay, Scott Foresman, 1979.

Pettigrew, T. F., The Ultimate Attribution Error: Extending Allport's Cognitive Analysis of Prejudice, Personality and Social Psychology Bulletin, vol. 5, pp. 461-476, 1979.

Pfeffer, J., and Salancik, G. R., The External Control of Organizations : A Resource Dependence Perspective, Harper & Row, 1978.

Porter, M., "How Competitive Forces Shape Strategy", Harvard Business Review, March/April, 1979

Raiffa, H., The Art and Science of Negotiation, Belknap/Harvard U. Press, 1982.

Rawls, J., Theory of Justice, Belknap/Harvard U. Press, 1971, 1999.

Rechtin, R., Systems Architecting: Creating and Building Complex Systems, Prentice Hall, 1991.

Rifkin, S., “The Parsons Game: The First Simulation of Talcott Parsons’ Theory of Action”, Ph.D. dissertation, George Washington University, 2004.

Royce, W. W, "Managing the Development of Large Software Systems", Proceedings, IEEE WESCON, 1970.

Scott Morton, M., The Corporation of the 1990s: Information Technology and Organization Transformation, Oxford University Press, 1991.

Scott, R., Organizations: Rational, Natural, and Open Systems, Prentice Hall, 2003.

Scott, W. R., Institutional Environments and Organizations, Sage, 1994.

Scott, W. R., Organizations: Rational, Natural, and Open Systems, Prentice Hall, 2003.

Shaw, M., Garlan, D., Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996.

Simon, H., Models of Man, Wiley, 1957.

Sommerville, I., Software Engineering, 5th ed, Addison Wesley, 1999.

Standish Group, Chaos Chronicles, Standish Group, 2004.

Sterman, J.D, Business Dynamics – Systems Thinking and Modelling in a Complex World, McGraw-Hill, 2000.

Swartz, N., "The Concepts of Necessary Conditions and Sufficient Conditions" (http://www.sfu.ca/philosophy/swartz/conditions1.htm), accessed 04/01/2007.

Thompson, J. D. Organizations in Action: Social Science Bases of Administrative Theory, McGraw-Hill, New York, 1967

Thorp, J., DMR's Center for Strategic Leadership, The Information Paradox: Realizing the Benefits of Information Technology, McGraw-Hill, 1998.

Torraco, R. J. , "Theory-building Research Methods", in Swanson, R. A., and Holton, E. F. III (eds.), Human Resource Development Handbook: Linking Research and Practice, pp. 114–137, Berrett-Koehler, 1997.

Treacy, M. and Wiersema, F., The Discipline of Market Leaders, Perseus Publishing, 1997.

Tversky, A., and Kahneman, D., "Rational Choice and the Framing of Decisions", in Decision Making: Descriptive, Normative, and Prescriptive Interactions, Bell, D., Raiffa, H., and Tversky, A. (eds.), pp. 167-192, Cambridge University Press, 1988.

von Neumann, J. and Morgenstern, O., Theory of Games and Economic Behavior. Princeton University Press, 1944.

Warfield J.N., A Science of Generic Design: Managing Complexity Through Systems Design, Iowa State, University Press, 1995.

Weinberg, R. S., "Prototyping and the Systems Development Life Cycle", Info. Syst. Management, pp. 47-53, 1991.

Wiest, J, D., Levy, F. K., A Management Guide to PERT/CPM, Prentice Hall, 1977.

Woodward, J., Industrial Organization: Behaviour and Control, Oxford University Press, 1970.

Wymore, A. W., A Mathematical Theory of Systems Engineering: The Elements, Wiley, New York, 1967.

Appendices

Appendix A: Analysis of UniWord

Background

This case study was published in (Boehm, 1981). Universal Micros, Inc. was developing a new product: An advanced personal computer named UniWindow. In order to make its product more appealing to its potential customers, Universal Micros decided to contract with several software houses to produce software packages for its new personal computer. One such package was the UniWord – a Word Processor for UniWindow that would be developed by the winning bidder to the Request for Proposal that was based on a set of high-level specifications provided by Universal Micros.

SoftWizards, Inc., a software contractor, was selected as the winning bidder among four other companies that had also bid for the contract. With some past experiences with developing text editors, Soft Wizards believed that winning this contract would provide a significant value addition to the company. Therefore, in excitement of this prospective opportunity, SoftWizards agreed to build the product with a much lower price tag than its competitors.

Within a few months into the project, problems began overshadowing success. Toward the expected completion date, SoftWizards failed to deliver the

product. This initiated a project audit by Universal Micros, which forms the basis of this post-hoc analysis.

VBSE Theory Analysis

Did the project identify its stakeholders' value propositions?

With a few exceptions, the project did not identify its stakeholders' value propositions. The project failed to identify all the stakeholders that were critical to the success of the product. As such, value propositions of unidentified stakeholders were therefore ignored. These stakeholders include product maintainers; Universal Micro's marketing staff; and proxy users representing the consumers at large. Each such stakeholder would have brought in value-propositions that could have, if not alleviate, provide the much required visibility and conformity to requirements. For example, proxy users would have identified poor interfaces, the marketing staff could have helped prioritize requirements based on minimal features required for a successful product exhibit, and the maintainers could have identified problems with vague specifications.

Did the project prioritize requirements?

There is insufficient evidence for a definitive answer, however it appears that more emphasis was placed on identifying the best technical approach for how the word processor should operate than on any prioritization activities (page 2,

paragraph 1). This assessment is also supported by the fact that there were no interactions between the clients and developers relating to prototypes, core capability demonstrations that usually form the thrust for reprioritizations (page 4, paragraphs 5-6). Further, when a schedule crisis became unmanageable towards the expected delivery date (page 2, paragraphs 11-12), SoftWizards elected to add new development personnel over reprioritization.

Did the project conduct expectations management?

No, the project operated on wishful thinking over pragmatism. First, any source selection based on lowest-cost-winner leaves little room for expectations management (except for cost/price) during project initiation. Second, while problems had started surfacing as early as the second month into the project, none of these indicators initiate any form of interaction between the client and developers until towards the end of expected completion date.

Did the project engage in prototyping?

As discussed before, there were no interactions between the clients and developers relating to prototypes, core capability demonstrations that usually form the thrust for reprioritizations (page 4, paragraphs 5-6)

Did the project identify its success-critical stakeholders?

Few. As explained before the project failed to identify all the stakeholders that were critical to the success of the product. As such, value propositions of unidentified stakeholders were therefore ignored. These stakeholders included product maintainers; Universal Micro's marketing staff; and proxy users representing the consumers at large.

Did the project identify its organizational dependencies?

There is insufficient evidence to make this assessment.

Did the project identify interdependencies between stakeholder values?

No. However, there are indications of interdependencies between stakeholder values that were neither identified nor resolved. First, few members in the development team had their values rooted in self-constructed theories for best way to develop the product. The impact of these conflicting values was large enough for key personnel to quit and to significantly deplete the team's morale (page 2, paragraph 4; page 4, paragraph 8). Second, clients placing value on cost and schedule, marketing staff requiring state-of-the-art features, and developers on getting the product to work in time and cost was another key interdependency that was overlooked.

Did the project identify product interdependencies?

Not until they surfaced on their own, however it was too late by then. The report cites numerous occasions where parts of the product refused to work with one another (page 2, paragraphs 2-4, 7; page 4, paragraph 1).

Did the project identify process interdependencies?

No. This is reasonably clear in many instances – insufficient test data and drivers available for testing when required (page 2, paragraph 2); vague schedule, abstract Gantt charts (page 4, paragraphs 4-6); and improper configuration management (page 4, paragraph 3).

Did the project negotiate a win-win product/process plan?

Doing source selection through lowest-cost-winner is usually not a win-win. However, even if we assume that was the case, having not identified all success-critical stakeholders or their win conditions destabilized the win-win equilibrium. For example, the development team's morale was very low; they didn't have job stability (page 4, paragraph 8) -- these are few indications that their win conditions were not factored in.

Did the project use risk to establish feasibility and commitment?

Perhaps one of the biggest shortcomings of the project was not having any form of risk management to help steer its direction. For example, measuring level of difficulty on certain product features either through prototyping or group assessments would have helped to determine its corresponding risk and thereby allowed decision makers to allocate resources appropriately and in some cases identify a fallback. In the case of this project all features were treated equal. Finally, a nine-month schedule to deliver a word processor with six developers also doubling up as designers and testers was overly ambitious. Schedule was perhaps one of their biggest risk factors, however, still, any risk management was missing.

Did the project adapt its plans to key changes in success-critical stakeholders' value propositions?

The most prominent change in value propositions of success-critical stakeholders was when the development team realized how far behind schedule it was. While it was too late to introduce new personnel into the team, credit is given for adapting to resource needs. However, what the project needed was rethinking its plans in terms of what's doable and what's not – basically expectations management and reprioritization in the form of simple software project planning.

Did the project have a business case?

There is insufficient evidence to make this assessment. While the project describes SoftWizard's winning bid (page 1, paragraph 9) ($300,000), and a profit sharing ($10 per license) element, the origin of these numbers, costs and projected benefits were not been discussed.

Did the project use stakeholder values for its product or process related decisions (initially or during changes)?

Both product and process related decisions were made without having any stakeholder values in context. Product decisions, for example, were made based on personal preferences of developers (page 2, paragraph 1-2), whereas process decisions were usually just based on project manager's insights.

Did the project establish a control mechanism to track progress?

High level Gantt charts and team members' weekly reports were the two tracking mechanisms used through the course of the project. Both the mechanisms however seemed of little use as they would stagnate at about 90% for each task (page 4, paragraph 6).

Did the project monitor and measure stakeholder value?

As explained above, high level Gantt charts and team members' weekly reports were the two tracking mechanisms used through the course of the project. Since they only tracked progress, the project clearly failed to monitor and measure most stakeholder values.

Appendix B: Analysis of MasterNet

Background

This case study was published in (Glass, 1998). In 1982, Bank of America, along with Seattle-First National, United Virginia, and Philadelphia National initiated the development of a state-of the-art trust accounting system. Having internally failed once in doing so and losing $6 million, the consortium of banks contracted Premier Systems to develop their trust accounting system that would manage their $38 billion portfolio in institutional and personal trusts.

Premier Systems was a relatively new company but its leadership, primarily Stephen Katz, had a strong background in financial systems and some success developing such systems through his previous company. With some initial research into the new system that the consortium of banks required, Premier began developing the banks' trust accounting system and promised delivery within 11 months.

After spending $78 million on development and losses incurred due to the new system's malfunctions, the banks finally handed their trust business to a subsidiary on account of inability to handle trust related services. Along with financial losses, plenty of bad press such as "$80 Million MIS Disaster" in newspaper headlines, the bank's image as a successful technological leader was negatively changed forever.

VBSE Theory Analysis

Did the project identify its stakeholders' value propositions?

Yes. The project orchestrated a very effective process in which a committee representing every affected stakeholder was constituted to identify their respective values. The committee met monthly to define requirements. Bank of America's staff met weekly with Premier's designers to discuss progress and additional needs. Also, expert users were brought-in from all banks to become involved with the design process and provide continuity from design to implementation (page 4, paragraph 1). Even the users' value propositions, such as comprehensive training programs, using videotape, classroom, and hands-on terminals, and voice in acquisition, was identified.

Did the project prioritize requirements?

As critical it is to identify stakeholder value propositions, it is equally important to prioritize requirements. Unfortunately, the MasterNet project miserably failed in prioritizing its requirements. This also resonated with what some executives internal to the bank felt – that is, the diversity of interests caused developers to accommodate all needs instead of limiting functionality (page 18, paragraph 1).

Did the project conduct expectations management?

While it is not clear if the project personnel engaged in any form of expectations management, certain events in the timeline of MasterNet suggest that doing expectations management was extremely critical to this project. For example, Premier Systems in its first presentation to the bank's executive board proposed a state-of-the-art system (built around multiple systems working together flawlessly) that featured unprecedented technology and promised wonders for their trust services. Further, in the presentation the entire MasterNet system, which eventually took five years and still was unfinished, and accumulated over 3.6 million lines of code was initially proposed as a mere 11-month minimal-risk project.

Did the project engage in prototyping?

Prototyping helps reduce risks – VBSE 101. In a project as big and technically complex as MasterNet, and with a $38 billion financial risk exposure, one would expect to see a tremendous amount of prototyping efforts guiding the way to success. However, neither Premier systems engaged itself in much prototyping, nor did Bank of America require Premier to establish technical feasibility.

Did the project identify its success-critical stakeholders?

Yes, as mentioned before the project receives extra credit for doing a wonderful job of identifying success-critical stakeholders. The project formed a committee representing every affected stakeholder in the participating banks. The committee met monthly to define requirements. Also, expert users were brought-in from all banks to become involved with the design process and provide continuity from design to implementation.

Did the project identify its organizational dependencies?

To some extent, yes, they were successful in identifying organizational dependencies. For example, the trust department was dependent on other departments such as the banks' securities and clearings departments. However, a few critical dependencies remained unidentified. For example, when the MasterNet project was initiated, the bank was undergoing a series of structural changes that shifted authority in places that had a direct impact on project's success (page 4, paragraph 1) and those changes were not taken into account!

Did the project identify interdependencies between stakeholder values?

There is insufficient evidence to make any conclusions about the project identifying interdependencies between stakeholder values, or resolving them amicably. However, it is clear that there were a few indications of conflicting interdependencies between stakeholder values. For example, Bank of America had always used IBM mainframes and wanted to continue using them as their choice of hardware – based on their preference for familiarity with IBM systems and previous positive experiences. On the other hand, Premier Systems preferred Prime's hardware over IBM. Some other conflicting interdependencies also emerged after Bank of America's corporate restructuring.

Did the project identify product interdependencies?

MasterNet was a huge and complex project that interfaced with many departments across various banks and organizations. As such product interdependencies are bound to exist. While the report does not cite any specific product interdependencies, joint meeting between the various departments in the course of design and analysis make the researcher believe that product interdependencies were identified.

Did the project identify process interdependencies?

Other than identifying deployment related processes such as training, data conversion, there is insufficient evidence to state if process interdependencies were identified.

Did the project negotiate a win-win product/process plan?

As mentioned before, there were a few indications of conflicting interdependencies between stakeholder values. For example, Bank of America had always used IBM mainframes and wanted to continue using them as their choice of hardware – based on their preference for familiarity with IBM systems and previous positive experiences. On the other hand, Premier Systems preferred Prime's hardware over IBM. Eventually however, much of these conflicts were resolved to establish a win-win. Unfortunately, the win-win equilibrium was destabilized many times as MasterNet coasted through rough seas – and as will be explained later there was little effort to restore the equilibrium.

Did the project use risk to establish feasibility and commitment?

The MasterNet project had a very strong commitment from all its success-critical stakeholders however; they failed to conduct any risk management or establishing technical and business feasibility.

While one objective of Bank of America was to also be able to sell the new trust system to other banks, the MasterNet project was essentially a "catch-up

with the world" project that bet its money on continuing status-quo rather than generating new revenue. This is further supported by the fact that Bank of America was still profiting $100 million a year through its trust services that were still running the vintage system (page 10, paragraph 1). Therefore, undertaking the risk of deploying a system that runs on 3.6 million lines of code and new hardware, within 11 months, and carries a financial risk exposure of $38 billion is perhaps the biggest indication risk imbalance.

As mentioned before, prototyping helps reduce risks. Much of MasterNet's failure was attributed to either software or hardware not scaling up to the needs of Bank of America's massive client base. However, neither Premier systems engaged itself in much prototyping, nor did Bank of America require Premier to establish technical feasibility.

Did the project adapt its plans to key changes in success-critical stakeholders' value propositions?

During the initial development phase there were no key changes in the success-critical stakeholders' value propositions. However, during the deployment of MasterNet problems relating to technical glitches and corporate restructuring hurt the team's morale at a very rapid rate. Even though financial relief through additional funding was provided by senior management, what the team actually required was job stability, innovative leadership and a plan that could set things

right (as observed by the research in various excerpts). For example, instead of having the teams continue putting extended hours into the conversion and deployment for the entire institutional trust business, they should have migrated only select customers while planning to migrate the rest incrementally.

Did the project have a business case?

MasterNet project was a "catch-up with the world" project that bet its money on continuing the status quo in ante rather than generating new revenue. Additionally, Bank of America also hoped that it would be able to sell its new trust system to other banks to break-even on their development costs. As such, the benefits were strong enough to warrant a new system however a strong business cases is certainly more than simply identifying the benefits and approving the costs proposed by its vendor. For example, a quantifiable business case helps facilitate product and process decisions. In the case of MasterNet, if Bank of America had perhaps sketched a 5-year plan with incremental releases of additional trust functionality, it would not only have limited its risk exposure but also created a platform for offsetting costs related to MasterNet through earnings on each incremental release.

Did the project use stakeholder values for its product or process related decisions (initially or during changes)?

There is insufficient evidence to make this assessment. However, since the overseeing committee representing the various departments actively got involved during product design and definition, it is assumed that they did.

Did the project establish a control mechanism to track progress?

MasterNet originally planned to be developed in 11 months, actually took five years and yet still failed. Setting milestones at regular intervals and establishing a control mechanism help stakeholders to check progress, reassess risk and adapt to changes/problems. However, in the case of MasterNet no milestones or any other effective controls were instrumented by the management. However, it is assumed that perhaps there was some ad-hoc tracking mechanism in place that is not documented.

Did the project monitor and measure stakeholder value?

Since the project did not have much of control mechanisms in place, stakeholder values were neither monitored nor measured.

Appendix C: Analysis of Windows for Word

Background

This case study was published in (Kemerer, 1997). Microsoft introduced its first word processor for the PC called the PC Word in 1983. Having been disappointed with lukewarm reviews and mediocre sales, Bill Gates in 1984 initiated the development of a new state-of-the-art word processor for its Windows operating system codenamed Cashmere (later changed to Opus).

Although its schedule slipped significantly from the originally projected ship date, Microsoft shipped Word for Windows with sales exceeding its own projections, and received significant critical acclaim press – it was Microsoft's first word processor to be rated higher than its competitor WordPerfect by InfoWorld.

VBSE Theory Analysis

Did the project identify its stakeholders' value propositions?

In today's world, most commercial products need to support and interface with a variety of third party products – all seamlessly working together. Microsoft's Word for Windows however was a commercial product for the users of Microsoft Windows in the 1980s and 90s. This meant, other than making the software work on Windows, and impressing customers with some really advanced features, the project was free from many of the constraints that have emerged in today's world.

This also means that the project then had few success-critical stakeholders involved in the project, and even fewer value propositions. While implicitly, their value propositions were identified.

Did the project prioritize requirements?

No, the project's development approach was very informal from today's standards and identifying which features to implement next was left open to the development team per their preferences.

Did the project conduct expectations management?

There is insufficient evidence to make any conclusions if the project took any initiative of managing expectations of its stakeholders.

Did the project engage in prototyping?

In terms of functionality and look-and-feel, Microsoft's Word for Mac and some initial research done by experts in word processors served as a good prototype for the development team.

As for prototyping for technical feasibility, Microsoft Word was required to run on Windows, which was also a Microsoft product. This significantly reduced the project's technical risks.

Did the project identify its success-critical stakeholders?

Yes, as mentioned before the project then had only a few success-critical stakeholders all of which were already involved in the project – these included the marketing, product and program management, user education and localization teams (page 707, paragraph 3).

Did the project identify its organizational dependencies?

It is not clear if the project had any organizational dependencies other than the fact that success of Windows as an operating system was tied to the success of Word. Word's success would also boost sales for Windows. On the flip side, its failure could hurt its sales as well.

Did the project identify interdependencies between stakeholder values?

Management's time to market vs. development team's need for flexibility and less pressure was perhaps the biggest interdependency between stakeholder values. While this was not explicitly identified in the project, it was still a well-known fact to all involved (page 709, paragraphs 1-4).

Did the project identify product interdependencies?

While the team created a set of formal specifications, product designs were extremely informal. This is clearly evident through the problems they had in fixing bugs, and through the amount of rework that went in adding features. Further, how to implement features was left up to the developers rather than having them adhere to any design.

Did the project identify process interdependencies?

In the beginning of the project, there was little if any process for developing Word. However, as the product started getting developed and made progress, some processes were instituted. In doing so, process interdependencies between development, testing, stabilization, localization were also identified.

Did the project negotiate a win-win product/process plan?

Much of development of Word was influenced by Bill Gates. He set features, while developers enjoyed freedom with technical aspects. Therefore other than schedule pressure on the development team, it was a natural win-win product/process plan (page 711, paragraph 7).

Did the project use risk to establish feasibility and commitment?

Given the facts that Microsoft Word was required to run on Windows, which was also a Microsoft product; Microsoft's success and experience with Word for

Macintosh; few competitors in the Windows world; Microsoft enjoyed a significantly low risk on the project. These factors were also good indicators of the project's feasibility.

Did the project adapt its plans to key changes in success-critical stakeholders' value propositions?

Yes, every new feature identified by Bill Gates and the senior management in their reviews was added to the list of features to be implemented.

Did the project have a business case?

Yes, as explained before developing a word processor for the Windows operating system was not only a low risk project for Microsoft but if successful, it would cater to a huge customer base while also boosting sales for its operating system.

Did the project use stakeholder values for its product or process related decisions (initially or during changes)?

Yes, all product related decisions were based on the features required and approved by the senior management (specifically Bill Gates). However, time which was also one of the key values was usually not factored in their decisions.

Did the project establish a control mechanism to track progress?

Yes. While it is not clear what forms of control mechanisms were instrumented, there is still evidence that the project had control mechanisms in place. The project's post-mortem report included statistics on estimations vs. actual (page 711, paragraph 5).

Did the project monitor and measure stakeholder value?

With respect to Microsoft Word, the key success-critical values were time, product quality and market-dominating features for the senior management, and development flexibility for the developers. Time, quality (in terms of identified bugs) and features were all monitored and measured. Development flexibility on the other hand is usually not measurable but the development team seemed to enjoy their freedom.

Appendix D: London Ambulance Service

Background

This case study was published in (Finkelstein, 1993). London Ambulance Service (LAS) was in the business of dispatching ambulances to individuals requiring urgent medical care. As such, when an emergency call is received by the LAS, ambulances are dispatched based on an understanding of the nature of the call and the availability of resources. In 1991, after LAS' management felt that an automated centralized ambulance service was essentially the way forward, it formed a small teams and assigned it the responsibility for generating a request for proposal (RFP), and thereby identify a contractor based on the meritocracy of each proposal submitted. Schedule and cost were the two most critical drivers for the team in identifying a suitable contractor.

After spending a few months developing an initial set of specifications for such an automated systems that formed the basis of LAS' RFP, the cheapest tender was accepted and a new system was developed and introduced the following year. The newly developed system however turned out to be a complete failure, resulted in a few casualties and was eventually rejected by LAS. Today, it is considered as a classic point of reference for "software disasters".

VBSE Theory Analysis

Did the project identify its stakeholders' value propositions?

With a few exceptions, the project did not identify its stakeholders' value propositions. The requirements specification document which formed the basis of the request for proposal was written by an analyst contractor, the systems manager and a few other individuals representing different departments. However still, having a few representative managers seldom provide a good estimation of the value propositions embedded in their respective teams. Moreover, as will be explained later, the project failed to identify all the stakeholders that were critical to the success of the product. As such, value propositions of unidentified stakeholders were therefore ignored. These stakeholders include ambulance crews and many other related departments (page 14, paragraph 3).

Did the project prioritize requirements?

No, the case study did not report any form of prioritization done either by LAS, or its contractors. As discussed before in Chapter 5, the project's approach was not iterative, rather the general belief was that the entire system would be developed in a single phase, and LAS will deploy the system in a cold turkey fashion. However still, there is evidence that while system capabilities were not prioritized,

identified defects were sorted in terms of their severity. And, priority was given to defects in the order of their severity.

Did the project conduct expectations management?

No. While LAS was actively involved in routine project management activities and well informed of many of the problems faced by contractors, there was no expectations management done by either side in setting more realistic time frames for CAD's development. Also, while some level of training was provided to the users of the new system, LAS should have also involved in setting the user expectations towards the new system. This would have helped alleviate some of the problems LAS had when users lost confidence in the system (page 3, paragraph 1).

Did the project engage in prototyping?

No. LAS did not invest any time in prototyping the system. With an extremely limited schedule, and little room for flexibility, the decision to not to prototype may have seemed appropriate to the development team. However, with the all uncertainty and risks the project embodied by using untested technologies and devices (page 22, paragraph 3), the theory would seriously raise a red flag for any project that chooses not to prototype.

Did the project identify its success-critical stakeholders?

Some, however the project failed to identify rest of the stakeholders that were also equally critical to the success of the product. These stakeholders include the general public, ambulance crews, trade unions, and many other related departments (page 14, paragraph 3).

Did the project identify its organizational dependencies?

No. As explained in Chapter 5, the LAS' CAD falls into the category of socio-technical systems that penetrate deeps into the human ecosystem and disrupts existing social structures. In the case of LAS, developing a computer-aided dispatch system would have significantly shifted power across many of its internal departments. None of these were accounted for. Additionally, LAS' dependency on RHA in providing guidelines for system acquisition resulted in LAS giving more emphasis on cost and schedule over contractor's capability in delivering such a system and past experience in managing risk. LAS was indeed a risky project, however due to its organizational interdependency with RHA lacked the capability to make this judgment early into the project.

Did the project identify interdependencies between stakeholder values?

No. The theory found that the project had very strong interdependencies between stakeholder values, and some of them were conflicting in nature. For example, LAS wanted Apricot Systems to prime the contract. Apricot instead felt that this was not in their favor as they did not have complete control over the project and therefore left the onus on SO to prime the contract. SO on the other hand felt that Apricot should prime the contract as it would relieve them of the pressures related to taking the lead. An another example involves users' limited enthusiasm with giving up control to an automated system, and also having to relocate per the needs of the new system disrupting their internal social structure.

Did the project identify product interdependencies?

Some. The study has limited evidence through a design document that shows how the various components of the product integrate with each other. However, based on some of the problems reported in difficulties in making the different systems talk together in a graceful fashion, it is concluded that only few of the many product interdependencies were identifies. This is also consistent with the limited time contractors had in designing the system.

Did the project identify process interdependencies?

LAS chose the PRINCE Project Management Method to guide the project towards success. Unfortunately, none of the stakeholders had any experience with PRINCE

and therefore lacked the capability to identify the many process interdependencies that are rooted in most software projects. The study also reported that due to the team's lack of direct experience with PRINCE, it was not sincerely followed through the course of the project (page 21, paragraph 3). This resonates with the problems the theory identified.

Did the project negotiate a win-win product/process plan?

No. LAS' planning was either based on policies it inherited from RHA, or driven by the technologies that Apricot and SO brought in. A win-win product/process plan is built around the value propositions of its stakeholders as opposed to policy or technology driven. Additionally, some planning elements were a direct outcome of bad guesses. For example, setting an inflexible timeframe of one year for a system as complex as CAD was essentially a bad guess which had no justification, or any room for negotiation.

Did the project use risk to establish feasibility and commitment?

Risk was perhaps the most underused (next to negligible) element in this particular project as though there was absolutely no amount of risk involved in developing an ambulance dispatcher service. Further, while commitment was established through inked contracts and management orders, feasibility like risk remained an anonymous element through the entire course of the project.

Did the project adapt its plans to key changes in success-critical stakeholders' value propositions?

No. The most convincing evidence to this assessment is the fact that LAS rather chose to deploy a buggy system against the wishes of many involved than reevaluate its priorities with respect to its current state of affairs.

Did the project have a business case?

Yes, LAS would have greatly benefited from a successful implementation of an automated ambulance dispatch system by not only meeting the ORCON standards emergency response times, but also in providing a better and more effective system in the interest of public welfare.

Did the project use stakeholder values for its product or process related decisions (initially or during changes)?

No. Product and process related decisions were either inherited from RHA, or technologically driven. An example follows: in spite of the team's lack of direct expertise with PRINCE Project Management Method, it was chosen as the guiding process for the project.

Did the project establish a control mechanism to track progress?

It is not clear if the project had established any control mechanism to track progress however, based on hindsight knowledge it appears that it indeed did not have any control mechanism in place.

Did the project monitor and measure stakeholder value?

No, stakeholder values were neither monitored nor measured. As indicated before, the project had severely failed in identifying stakeholder values, as such there was nothing to monitor and measure.

Appendix E: CMU Surface Assessment Robot

Background

This case study was published in (Latimer, 2007). The study examines a project undertaken by CMU as a contractor to build a robot that can inspect the smoothness of a road surface while maintaining the same inspection quality as the manual method historically employed. While the project successfully delivered a robot that was designed right to the specifications, and if made operational would have also satisfied the estimated return on investment, it was however not transitioned into operational use.

VBSE Theory Analysis

Did the project identify its stakeholders' value propositions?

Yes, the project identified all of its success-critical stakeholders (except for one that emerged later in the project – the acquirers of the acquiring organization) and their value propositions. Through all the phases of the development, success-critical stakeholders such as engineers, managers, quality assurance personnel had come together in identification of key propositions, and in reviewing progress with respect to set milestones (page 38, paragraph 2; page 32, paragraph 6).

Did the project prioritize requirements?

While the documented case does not explicitly support this conclusion, statements such as “verify and validate the requirements to ensure no “gold-plating” of requirements” (page 33, paragraph 3), along with efforts invested towards prototyping are indicative of prioritization. Further, personal communication with the author of this case who was also involved in the project verified that requirements prioritization was done in multiple review sessions through the course of this project.

Did the project conduct expectations management?

Yes, stakeholders from the acquiring organization along with other critical stakeholders consistently participated in concept formation, requirements, design and reviews. Project’s current progress and future goals were effectively communicated to all of them.

Did the project engage in prototyping?

The project frequently engaged in prototyping. Many requirements were validated through initial prototypes, thereby also establishing technical feasibility for the project (pages 32-33).

Did the project identify its success-critical stakeholders?

Yes, as mentioned before the project identified all of its success-critical stakeholders (except for one that emerged later in the project – the acquirers of the acquiring organization) and their value propositions. Through all the phases of the development, success-critical stakeholders such as engineers, managers,

quality assurance personnel came together in identification of key propositions, and in reviewing progress with respect to set milestones (page 38, paragraph 2; page 32, paragraph 6).

Did the project identify its organizational dependencies?

No, and this unfortunately contributed as the primary reason for the project's failure. The acquiring organization of this robot was also being acquired by another bigger organization that preferred to buy well-established off-the-shelf products over developing them in-house. According to the VBSE theory, identifying organizational dependencies is critical to project success because along with new stakeholders, they also bring in process/product/value dependencies that often conflict with existing structure. Further, in the case of this project, this new organizational dependency had created a fundamental shift in organizational control.

Did the project identify interdependencies between stakeholder values?

While there is no evidence in the documented case to make any conclusion, there is no evidence to the contrary as well. In hindsight however, with the new organizational dependency that had formed in the organization, there were indeed a few interdependencies between stakeholder values.

Did the project identify product interdependencies?

Yes. The project had devoted a lot of effort towards tracing dependencies between requirements and design elements across the various design phases (page 34, paragraph 6).

Did the project identify process interdependencies?

Yes, project processes such as milestones were carefully planned. For example, the project had established a timeline (page 32, paragraph 5) for various project phases, each phase had its associated set of goals and exit criteria that was reviewed consistently and continuously.

Did the project negotiate a win-win product/process plan?

Yes, as mentioned before, the success-critical stakeholders periodically met to negotiate and decide a win-win product and process. In certain situations, for example in the preliminary design phase, when a particular solution was disliked by a few stakeholders, it was always documented and acted upon either by revisiting the requirements, or adding constraints to existing plans.

Did the project use risk to establish feasibility and commitment?

Yes. Risk was a constant factor in making project-related decisions. For example, all requirements were evaluated based on their level of risk. Low-risk requirements/technologies were not prototypes. Emphasis was given to the requirements that carried significant risk, and commitment or feasibility was either established through prototyping or group's consensus on accepting risk.

Did the project adapt its plans to key changes in success-critical stakeholders' value propositions?

No. As explained before, the project was unable to identify its organizational dependencies and thereby factor in new stakeholders and their value propositions.

Did the project have a business case?

Yes. The project had a very strong business case. Their projected return on investment was in the order of 1:100 (page 28, paragraph 3). The to-be-developed system (robot) would have replaced an existing manual process the organization was using in assessing road surfaces.

Did the project use stakeholder values for its product or process related decisions (initially or during changes)?

Yes. As discussed before, risk, personal dislikes and other stakeholder values were used to steer product and process related decisions. Prototyping was based on stakeholders' comfort level and acceptable risk, design decisions were made in alignment with engineers' past experiences and preferences.

Did the project establish a control mechanism to track progress?

Yes. The team had weekly meetings to track and discuss progress. Further a project management plan, and a life cycle management was also created which helped steer the course of development (page 32, paragraphs 2-3).

Did the project monitor and measure stakeholder value?

Yes. Each implementation unit was mostly derived from an existing prototype was traced to a design document, and a requirements document. The requirements represented a good share of stakeholder values. Further, the life cycle plan, project management plan, design reviews, and weekly meetings served as a way to orchestrate project planning, monitoring and control.