SELECTING A SOFTWARE ENGINEERING METHODOLOGY USING MULTIOBJECTIVE DECISION ANALYSIS

THESIS

Scott A. O'Malley, First Lieutenant, USAF

AFIT/GCS/ENG/01M-08

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

SELECTING A SOFTWARE ENGINEERING METHODOLOGY USING MULTIOBJECTIVE DECISION ANALYSIS

THESIS

Presented to the faculty of the Graduate School of Engineering and Management

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Scott A. O'Malley, B. S.

First Lieutenant, USAF

March 2001

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense or United States Government.

SELECTING A SOFTWARE ENGINEERING METHODOLOGY USING MULTIOBJECTIVE DECISION ANALYSIS

THESIS

Scott A. O'Malley, B. S.
First Lieutenant, USAF

Approved:

Maj Scott A. DeLoach, Ph.D. Chairman	26 Feb 01 Date
Dr. Thomas C. Hartrum Committee member	2/26/01 Date
Capt Scott M. Brown, Ph.D. Committee member	26 Feb 01 Date

ACKNOWLEDGMENTS

I would like to take this opportunity to express my most sincere appreciation to my faculty advisor, Major Scott DeLoach, for his guidance and support throughout this endeavor. His experience and insight kept me on the path to success, and his expectations forced me to set my goals high. I would also like to express my appreciation to the other members of my thesis committee, Dr. Thomas Hartrum and Captain Scott Brown, for their support throughout this process.

My biggest thanks goes to my wife, Angela, who has put up with me through the tribulations and shared my joys at the completion. The last eighteen months have been by far some of the best in our time together. I would also like to say thanks to my girls, who cannot read this yet, but someday may find this work to be inspiration to pursue their own dreams!

Scott A. O'Malley

TABLE OF CONTENTS

TABLE OF FIGURES

ABSTRACT

With the emergence of agent-oriented software engineering methodologies, software developers have a new set of tools to solve complex software requirements. One problem software developers face is to determine which methodology is the best approach to take to developing a solution. A number of factors go into the decision process. This thesis defines a decision making process that can be used by a software engineer to determine whether or not a software engineering approach is an appropriate system development strategy. This decision analysis process allows the software engineer to classify and evaluate a set of methodologies while specifically considering the software requirement at hand.

The decision-making process is developed on a multiobjective decision analysis technique. This type of technique is necessary as there are a number of different, and sometimes conflicting, criterions. The set of criteria used to base the decision was derived from literature sources and validated by an opinion survey conducted to members of the software engineering community. After developing the decision-making framework, a number of case studies are examined.

[THIS PAGE LEFT INTENTIONALLY BLANK]

SELECTING A SOFTWARE ENGINEERING METHODOLOGY USING MULTIOBJECTIVE DECISION ANALYSIS

I. Introduction

The Air Force is changing the way it leverages its information technology through a strategy called "One Air Force…One Network" [33]. *Information superiority*, a key factor to success in the 21st century, is clearly delineated by visionary documents such as *Joint Vision 2020*, and *Air Force 2025* [17, 35]. Information superiority provides the ability to control and exploit information in a way that ensures decision dominance; that is, the ability to make smarter decisions faster than the enemy. The strategy is based on leveraging the advantages of current and future information technologies.

The technological advances in information systems have made information technology one of the military's greatest force multipliers. The ability to rapidly disseminate large amounts of data across geographically separated command centers allows commanders to make timely decisions in an ever-changing environment. As the ability to rapidly disseminate information increases, the amount of information available to decision makers is also increasing dramatically. It is imperative that technology advances to ensure that the information available is a benefit to leaders and not a hindrance due to volume. The size and complexity of such a worldwide network will necessitate formal and rigorous approaches to ensuring the entire system will be interoperable and secure.

The Department of Defense, through the Defense Advanced Research Project Agency (DARPA), is investigating the use of agent-based systems as a technology for maximizing the benefit of the information resources, while trying to reduce the strain placed on other resources, namely manpower and time [8]. The current focus has been on integrating client/server information systems together to provide decision support capabilities to the military and national leaders. Agent-based systems are a promising technology in this regard, as they offer benefits such as integrating legacy systems with newly developed

components, handling errors gracefully, moving tasks that involve simple reasoning but require a large amount of computation to the agent to allow the human more time for complex decision making, and allowing a synergistic collaboration between users and software agents.

As agent-based technology emerges, the question of how to apply this advancing field is raised. Through the Air Force Office of Scientific Research (AFOSR), the Air Force is investing in the development of agent-based technology to improve command and control support. Developing agents, which can be used to gather and synthesize data as well as cooperate with other agents to solve problems, is just one area on which AFOSR intelligent agent research is focused.

Current AFOSR sponsored research efforts at AFIT have involved defining a methodology, called Multiagent System Engineering (MaSE), for designing, implementing and verifying multiagent systems [20, 32, 39]. MaSE is an extension of object-oriented software design methodologies, where an agent is the basic building block of a system. The agent is an abstraction that encompasses many of the characteristics of a multiagent system. While developing this methodology, AFIT researchers have also developed agentTool, a multiagent system development tool that implements the MaSE methodology. Currently, the methodology is based on the assumption that an agent-oriented system is to be developed versus a traditional system [39].

This research examines the decision-making process, which leads to the decision to use a specific methodology, such as MaSE, to develop potential software projects. Agent system technology offers a number of advances over conventional software systems for the military's information sharing/decision support domain. Understanding these capabilities is central piece to deciding whether or not the technology is appropriate to particular problems. The next section presents a foundation of agent technology information.

1.1 Multiagent Systems

The Air Force's challenge to create a seamless network to ensure information superiority introduces an extremely complex operating environment. Decisions regarding network connectivity and

data storage are just a couple of the hardware factors in the task. The development of software around the hardware resources is another critical piece to the success of "One Network…One Air Force."

Multiagent systems are an attractive solution to the software problems. A *multiagent system* is comprised of many heterogeneous, or different, agents. An *agent* is defined as a computer system that is situated in an environment and is capable of autonomous action in the environment in order to meet its goals [40]. Agents are characterized as being autonomous, pro-active, reactive and sociable. *Autonomy* is the ability to act without outside intervention, as well as having control over internal state and behavior. *Pro-activity* is the ability to exhibit goal-directed behavior by initiating actions that work toward satisfying a goal. *Reactivity* is the ability to perceive the environment and respond to changes in the environment in order to achieve a goal. Finally, *social ability* is the ability to interact with other agents in order to achieve a goal. These properties allow agent systems to be flexible problem solvers.

Designers of the Air Force's global network must determine the type of environment the system will support. By considering two different environments—closed and open—properties of agent behavior can better be evaluated. In a *closed* system environment, the system designers know exactly what type of agents will exist. For the most part the agents will be *cooperative*, that is, the agents will work together toward a common goal. In an *open* system environment, agents are likely to meet both cooperative agents and competitive agents. *Competitive* agents are self-interested; they have their own set of goals, which may be in conflict with the overall goal of the system. Self-interested agents will assist with the group's problems, usually at some cost, which is negotiated prior to performing any computation. For an agent to work within an open system, the agent also must know how to handle communications with agents that were unknown to the system designer when the agent was developed [9].

The global nature of the "one network" concept typifies the problem domains that multiagent system approaches are suited to solve. That is, the problem domains generally have an inherent form of distribution; that is, knowledge, capability, information, and expertise are all resources that can be distributed throughout the system [3]. Three examples of problem domains with inherent distribution are distributed situation assessment, distributed resource scheduling and planning, and distributed expert

systems [23]. MaSE and other agent-oriented software engineering methods are well suited for dealing with distribution in software requirement problems.

By building systems that involve multiple agents, the system should realize benefits such as speed-up due to concurrent processing, less communications bandwidth requirements as information gathering is performed at the source and not remotely, and increased reliability since there is not a single point of failure. Other benefits include increased responsiveness due to processing and sensing being performed at the source and simplified system development, since each agent is modular [23].

1.2 Problem Statement

A number of methodologies exist for building multiagent systems, once the decision to use a multiagent design is made [11]. Like MaSE, these methodologies provide no guidance as to which types of software problems the methodology is suited. The academic community, as well as industry, is still trying to determine which problems call for a multiagent approach [15, 24].

Jennings and Wooldridge have written many papers on agent-oriented software engineering in which they provide "intellectual justification" [13,15] for the validity of the agent-oriented techniques. Their justification, however, comes from a qualitative analysis of how well the technique addresses the principles that allow software engineering techniques to deal with complex problems proposed by Booch: abstraction, decomposition, and hierarchy [2, 15]. They leave "understanding of the situations in which agent solutions are appropriate" as an outstanding issue [15].

In 1999, the European Institute for Research and Strategic Studies in Telecommunications (EURESCOM) began a project to explore the use of agent technologies within the European telecommunications industry. One of the project's three objectives is to "define guidance for the identification of application areas where an agent-based approach is better suited than other approaches" [24]. Below are the five guidelines the consortium has promoted that are intended to help the developer in deciding whether or not an agent-oriented approach is appropriate [24]:

1. An agent-oriented approach is beneficial in situations where complex/diverse types of communication are required.

2. An agent-oriented approach is beneficial when the system must perform well in situations where it is not practical/possible to specify its behavior on a case-by-case basis.

3. An agent-oriented approach is beneficial in situations involving negotiation, co-operation and competition among different entities.

4. An agent-oriented approach is beneficial when the system must act autonomously.

5. An agent-oriented approach is beneficial when the system is expected to be expanded or modified or when the purpose of the system is expected to change.

These guidelines are a beginning to determining whether or not an agent-oriented approach is well suited to the problem. However, based on these guidelines alone, there is still no clear answer as to whether or not the approach is appropriate. The problem this research seeks to solve is to *define a decision making process that can be used by a software engineer to determine whether or not an agent-oriented software engineering approach is an appropriate system development strategy.*

1.3 Problem Elaboration

With the problem statement in mind, additional topics require explanation. First, "*define a decision making process*" indicates that a working knowledge of decision theory is needed. Then, "*determine whether or not an agent-oriented software engineering approach is an appropriate system development strategy*" requires not only an understanding of agent-oriented software engineering methods but other software engineering techniques as well. This section provides information on both of these topics.

1.3.1 Decision Making

A goal of this research is to specify a decision-making process for determining an appropriate software engineering technique. Clearly, the decision to select one methodology over another is a difficult task where many trade-offs must be made. As such, a multidimensional technique for decision-making is required.

1.3.1.1 Design Space and Rules

The intention of the *Design Space and Rules* methodology is to assist new engineers in making the correct choice in design, as would an experienced software engineer [21]. This technique is the foundation for a decision-making process for software engineers when trying to determine which design options will provide a good solution to a given requirements statement. The *design space* is a multidimensional space for classifying systems. Each dimension of the design space represents different structural and functional characteristics of the system. Structural dimensions represent characteristics that pertain to the techniques that can be employed, whereas the functional dimension captures the impact of the decisions. As correlations between different dimensions are discovered, *rules* can be generated to assist in the decision process. The rules are functions that map one dimension to another. By establishing a set of rules, trade-offs between dimensions can be evaluated.

An example of two dimensions is response time (functional) and interprocess synchronization (structural). A point in the design space then determines a design. This small design space is illustrated below in Figure 1 [21].

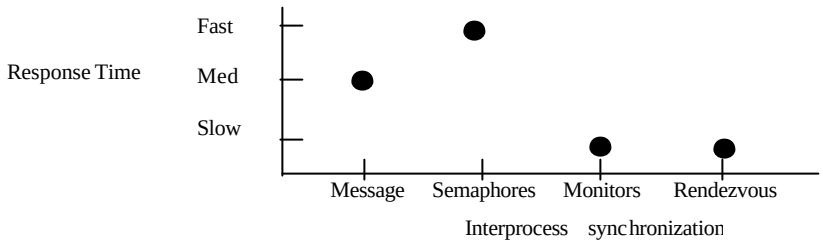


Figure 1: A Simple Design Space

A correlation that can be drawn from the simple design space in Figure 1 is the relationship between interprocess synchronization and response time. If the specification required a highly responsive system, then messaging or semaphores would be better design choices that monitors or rendezvous. Another functional dimension that the "Interprocess Synchronization" may be plotted against is complexity. The space may show that messaging and semaphores are "very complex." The software engineer is now required to balance the complexity of the system with the responsiveness. By testing

existing systems against the design space and rules, the degree of agreement—that is how the actual implementation compares to the design predictions—can be measured.

1.3.1.2 Multiobjective Decision Analysis

The field of Operations Research began looking at scientific approaches to decision making during World War II [6]. In the past several decades, increased awareness of the need to identify and consider many objectives simultaneously has led to research expanding the systematic procedures for making decisions based on a single criterion. From this field of study comes multiobjective decision analysis. Multiobjective decision analysis is based on a utility assessment framework, where alternatives are compared based on a set of the problems characteristics called *evaluation considerations*. A *score* is calculated for the individual evaluation considerations, which are then combined in order to derive an overall score of utility, or satisfaction. When combining the evaluation considerations' scores, a *weighting factor* is used to express the decision maker's rating of importance for the characteristic. The scores and weights are combined by a *multiobjective value function*. The multiobjective value function returns a fitness rating for an alternative. When compared, the alternative with the highest score is an optimal solution if the definition of optimal is relaxed to be "one that maximizes a decision maker's utility (or satisfaction)" [16].

1.3.2 Software Engineering Paradigms

Over the last four decades, software has become an important piece of information systems. During this time, software engineering paradigms have been designed as methodologies for creating good software in a reasonable amount of time and at a reasonable cost. These paradigms are generally layered approaches to developing solutions from a customer's non-technical description of what the system must be capable of doing. IEEE defines software engineering as

> *(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1). [10]*

Following a disciplined approach to creating software is the best way to ensure quality software is produced. Three such approaches are discussed below, as they are examples of emerging approaches to software engineering. The three paradigms are the object-oriented paradigm, the component-ware approach, and agent-oriented software engineering.

1.3.2.1 Object-Oriented

Of the three paradigms presented, object-oriented is the oldest. Object-orientation deals with the complexity of large software problems through the concepts of decomposition, abstraction, and hierarchy. Object-oriented software engineering assists the software engineer with the task of addressing the concepts of modern software engineering: information hiding, data abstraction, encapsulation, and concurrency. These concepts are not easily handled with the structured analysis techniques developed prior to object-orientation. Object-orientation also promotes the concept of software reusability. As the functions and attributes of the entity are encapsulated within the object, the object is easily migrated to other systems for reuse. An object-oriented application is a closer representation of the real world than a functional application. The ability to better understand the mapping of the real world to the software abstraction greatly improves the users confidence in the system. Additionally, the ability to understand the code is greatly enhanced by object-orientation, which in turn leads to better maintenance and improved modifiability. Since objects provide a certain level of data abstraction and information hiding, the underlying functionality of the object can remain the same, but the implementation can easily be changed to provide improved performance or additional functionality.

1.3.2.2 Component-ware

The component-ware paradigm is an attempt to maximize software reuse. Components are defined to be “physical, replaceable parts of a system that packages implementation and provides the realization of a set of interfaces” [19]. Components usually perform a single function and are treated as "black boxes" β4]. The original creation of components can come from any software engineering technique, such as object-oriented or structured analysis. Internally, the method of storing data is irrelevant to the user. Any information that is maintained by the component is only accessible by prescribed

operations provided by the component. The versatility of components is, therefore, great. The driving purpose behind developing and using components is software reuse. Systems are built by combining components. This component composition requires connecting interfaces. Components developed in an object-oriented method may provide for inheritance, thus allowing the system to make use of the methods within the component. Functionally developed components make use of parameterized function calls. In some instances, a scripting language may be needed to act as the *component glue* [37]. When a requirement arises for a new system, it is unlikely that a complete system meeting all of the requirements will be able to be generated. The Unified Modeling Language has been expanded to include representations for components. In many of the object-oriented methodologies that have expanded to include the incorporation of components, the decision to use a component is left to the design phase [19].

1.3.2.3 Agent-Oriented

Agent-oriented software engineering is an emerging paradigm based on the object-oriented paradigm. Like the object-oriented paradigm, the agent-oriented paradigm is designed to reduce the complexity through decomposition, abstraction and hierarchy. As expected, the primitive building block in the agent-oriented paradigm is an agent. Agents are more expressive than objects. Every agent is autonomous, having control over its own internal state and behavior and having well-defined boundaries and interfaces. By adopting an agent view of the world, it is apparent that most problems require multiple agents to represent some decentralized nature of the problem, multiple control centers, or multiple goals [42]. One difference between agents and objects is their interaction techniques. Objects interact with one another through method invocation, whereas agents communicate through a high-level agent communication language (ACL). ACLs allow the interactions to be conducted at the knowledge level. By using an ACL, agents do not need to know anything about the structure of other agents. The only requirement is that the agents know how to pass and accept messages from other agents.

1.4 Scope of Research

To better define this problem it will be necessary to focus the research on a limited number of software engineering techniques. Of course, those techniques should be mainstream in order to provide

enough information regarding their potential benefits as well as their pitfalls. For the development of agent-oriented software, the MaSE methodology will be used. On the other hand, a general object-oriented methodology using the Unified Modeling Language will be used for the object-oriented and component-ware software development.

1.5 Approach Overview

The remainder of this thesis describes a systematic approach to determining an appropriate software engineering methodology for a given software problem. The steps taken are summarized below:

1. *Determine a decision-making process* – This process is the framework by which the software engineer determines an appropriate technique to develop a given software problem.

2. *Define a decision analysis tool* – With the process selected, the definition of a tool based on the process is needed to evaluate different alternatives.

3. *Apply tool to software problems* – The tool is applied to a number of requirement specifications. Additionally, one of the requirements is developed using the two highest rated methodologies in order to collect data to validate the decision.

4. *Analyze programs* – The developed programs are compared based on a set of predetermined software metrics. Based on these metrics, a correlation is made between the actual software results and the results of the decision analysis tool.

1.6 Thesis Overview

Chapter II describes the approach taken to define a strategic decision making approach to problem analysis. Chapter III provides a decision analysis tool that can be used to determine the fitness of a multiagent system solution. Chapter IV provides a demonstration of the decision analysis on a number of systems. In addition to the application of the tool on the system specification, results of metrics collected during the development of one of the specifications are analyzed. Chapter V closes the thesis with

conclusions and future work. Additional support material is provided in the appendices. Appendix A is a literature review providing greater detail on many topics discussed throughout the thesis. Appendices B and C contain results of a survey conducted to validate the selection of factors used in the decision-making process, and sensitivity analyses on data collected on a set of use cases, respectively. Appendix D is a user's manual for a software application developed to implement the decision analysis process. Appendix E is a collection of methodology by-products and the results of a questionnaire used for the validation of the decision analysis tool.

II. Problem Approach

2.1 Introduction

A complex problem that software engineers encounter is the choice between engineering approaches. There are many software methodologies that software engineers can use to develop solutions to complex software problems. If quantifiable data that could be used to compare the development of similar projects existed, the data could be used to base decisions on which methodology to choose. Unfortunately that data does not exist. Leaving fewer quantifiable reasons, the engineer relies on subjective criteria to select one methodology over another, such as personal preference or expertise, management, etc. One alternative is to use the most familiar method without considering other alternatives. There is some credence to this approach since the organization understands this approach as it is used in the day-to-day business of software development. The argument to this choice, however, is that there may be a better way of solving the problem, as one tool does not generally fit every problem.

This chapter describes an approach to developing a tool for assisting a decision maker with the selection of the best alternative among many "good" alternatives that exist. The problems that the software engineer will face will generally have many different, and possibly conflicting, objectives; the output of this tool is not to report a definitive answer to the question of which software paradigm to use, but rather to provide an indication of good methodology candidates.

2.2 Method Selection

As described in Chapter I, the intent of this research is to specify a decision-making process for determining an appropriate software engineering approach. When faced with the choice of selecting an appropriate methodology, many factors effect the decision. Clearly, it is a difficult task where many trade-offs must be made. The challenge of balancing conflicting objectives and analyzing trade-offs between different characteristics is key to making a good decision. Multidimensional decision techniques provide the systematic procedure for handling the many facets of the methodology selection problem.

The main focus of research into decision-making techniques comes from the field of Operations Research; in fact, Operations Research was developed as a scientific approach to decision-making in military scenarios during World War II [6]. Some of the mathematical frameworks that have come out of this field of research are linear programming, inventory control, dynamic programming, Bayesian analysis, and simulation techniques [6].

Although many systematic procedures and mathematical frameworks for decision making exist, two techniques—one that was developed from a software architecture point of view, and the other a more traditional Operations Research technique—were explored during background research. The first technique, Design Spaces and Rules, was developed as a tool to provide software engineers architectural design guidance [21]. This focus on design phase issues takes functional requirements of the project and maps them to structural design choices. For example, a functional requirement for data storage could include a database management system or a flat text file as the structural realizations for the system. Correlating the different structural/functional mappings makes up design rules. The ability to objectively specify design rules would be dependent on the availability of performance data. Given appropriate data, this technique would be able to generate a tool for the problem of selecting what type of system to design, i.e. object-oriented or agent-oriented, but this decision would be based on mainly performance issues of certain design choices.

Determining an appropriate methodology based on the predicted performance of the software, however, does not really address the issue with regard to the methodology. Referring back to the data storage example, the type of methodology selected has no true bearing on the appropriate implementation. The real question for the methodology is how does it handle the representation of data storage, and is it handled in such a way that it is clear to the system designer? So, even if the data required to quantify these performance issues did exist, the decision maker would be disregarding many other factors vital to the decision.

This technique is also absent of any type of mathematical framework to back up any decision made by the tool. Again, had a large set of data been available and uncertainty factored in, it might be

possible to present a convincing argument. This data is not available, however, to argue the results effectively.

The second approach explored was Multiobjective Decision Analysis. Multiobjective Decision Analysis, an operations research technique, is derived from the utility assessment framework. For a problem like methodology selection, in which a finite set of alternatives is evaluated, utility assessment is considered superior to other mathematical frameworks such as mathematical programming [6]. Based heavily on decision theory, the technique does not require its users to know decision theory.

Unlike the Design Space and Rules technique, Multiobjective Decision Analysis allows the decision-maker to take issues other than performance into account, such as the impact that the decision will have on the current set of business rules. A summary of the factors that impacted the choice of a strategic decision-making technique is given in Figure 2. With these considerations in mind, Multiobjective Decision Analysis was selected for this research.

Factor	Design Space and Rules	Multiobjective Decision Analysis
Handling Multiple Criteria	✍	✍
Mathematical Framework		✍
Flexible	✍	✍
Large Body of Reference Available		✍

Figure 2: Factors for Decision Technique Selection

2.3 The Approach

To make use of the Multiobjective Decision Analysis approach to decision making, many details of the problem at hand had to be specified. Figure 3 depicts the approach taken to solve the paradigm selection problem in the general form.

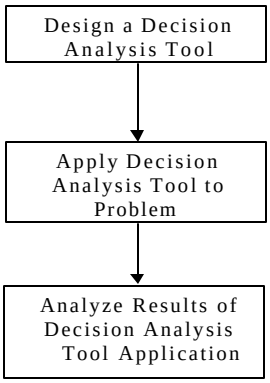


Figure 3: Problem Approach

2.3.1 Decision Analysis Tool Design

After selecting a decision analysis technique, the next step was to develop a decision analysis tool to assist the decision maker. There are a number of distinct steps that go into creating a decision analysis tool. The tool itself is realized through software; the use of a software package allows for repeatability and accuracy. The design of the decision analysis tool for this problem is discussed in Chapter III. The general steps taken are discussed below. Figure 4 shows the process that is followed in creating this tool.

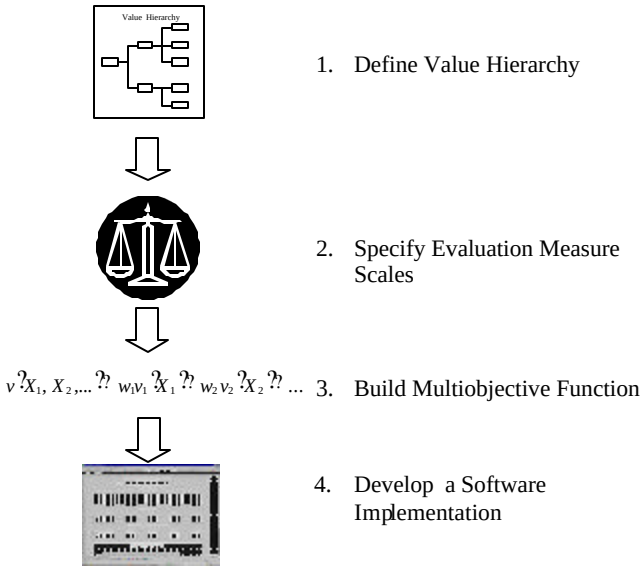


Figure 4: Decision Analysis Tool Development

2.3.1.1 Defining the Value Hierarchy

The first step in developing a decision analysis tool is defining a value hierarchy. A value hierarchy, which is a value structure that captures the objectives of the decision, is used to capture the factors that are important to the decision. This step is also one of the most critical steps in the process, because it specifies the issues that the alternatives must address in order to be viable solutions. If the value hierarchy is not complete, the decision maker will be making a decision that may ignore vital information.

The value hierarchy provides a graphical depiction of the evaluation considerations relevant to the decision. As the name suggests, a value hierarchy is organized in such a way that the evaluation considerations near the root of the tree are made up of the evaluation considerations that are its descendants. Formalizing the value hierarchy provides the decision-maker with a model representation of the factors going into the decision. This is beneficial to the decision-maker, who may have to explain or defend the decision to other interested parties.

The value hierarchy captures the factors of the decision that are important to the decision maker. Since this work is proposing a decision analysis tool that can be used for general software engineering methodology selection problems, the values, or considerations, must be determined for the general problem. The considerations selected were determined, first, from characterizations in current literature, and, then, validated by a survey of software engineering practitioners in academics, industry, and government.

Existing work in classifying software methods has described three major areas of characterization [5]. This classification scheme has been applied to the problem of selecting software engineering methodologies for the development of systems in the real-time system problem domain [38]. The process involves determining what a method is, what a method does, and what issues the method addresses. The three areas of classification are:

1. *Technical Characteristics* – This category looks at classifying the technical characteristics of the software development through the three stages of development

(specification, design, and implementation). The characteristics of the software problem that are dealt with during the specification—or analysis—phase relate to the behavioral and functional views of the problem. These views are carried through to the other stages of the system development. During the design phase, the behavioral and functional views are mapped into the behavioral and functional characteristics of the function. Effective methods allow for smooth transition across these stages and allow the ability to trace functional and behavioral characteristics through all stages of development.

2. *Management Characteristics* – It is important to consider the support that a method provides to management when evaluating different methods. The characterization should consider how well the method deals with the typical management and project issues such as estimating, planning and review. The characterization should also look at how the method is related to the needs and processes that exist within the organization. Management practices are often difficult to change and, therefore, identifying potential changes is an important factor in adopting a new methodology [5].

3. *Usage Characteristics* – Capturing and describing the characteristics of the method that will affect its use by an organization is also an important factor in evaluating and comparing methodologies. These characteristics include the basis for the methodology, the availability of training, and the availability of tool support. This characterization is important in understanding the magnitude of change involved with selection of a methodology.

With these characteristics in mind, a set of values was developed for the methodology selection problem. Rephrasing the initial problem of selecting an appropriate methodology for a particular software project, the root of the value hierarchy is the question "what is a good software engineering methodology that my organization can use to reduce the development costs and produce a quality product?" This question is at the heart of almost every software engineering paradigm and guides most of the current research in Software Engineering.

From this question, two distinct evaluation considerations can be identified. The first is the costs associated with the new methodology and the second is the quality, or how well the methodology can provide for the specific issues of the software problem. Costs, in this context, include not only monetary impact but also other types of impact on the organization and its existing processes. The three characteristics of software methods relate to these evaluation considerations. The management and usage characteristics belong to the area of *management issues*, where issues such as costs and the effects that the methodology has on the organization are key. Technical characteristics, those that deal with the functional, behavioral and structural view of the problem, reflect the quality evaluation consideration, *project requirements*. This layer of evaluation considerations is, next, refined.

These refined values were presented in a survey to software engineering professionals. The survey requested that the professionals provide their expert opinion on the relevance of the considerations being proposed. Figure 5 shows a partial example of the value hierarchy developed for the methodology selection problem.

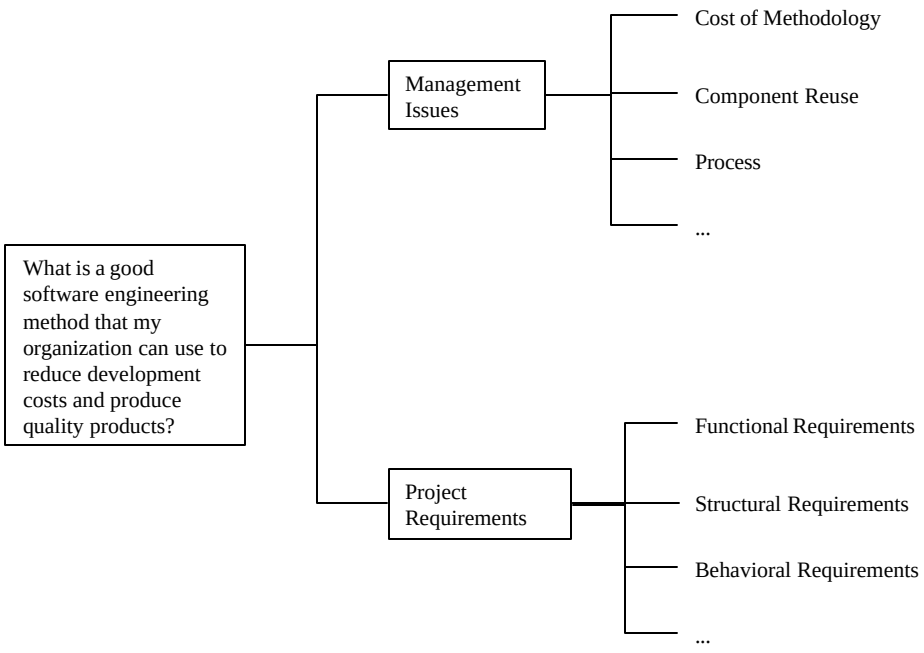


Figure 5: Example of a Value Hierarchy for Methodology Selection

In addition to specifying the evaluation considerations, objectives are determined. With most evaluation considerations, one will either attempt to maximize or minimize the consideration. Maximizing is used in the case where the alternative is a benefit to the evaluation consideration; on the other hand, minimizing is used where the evaluation consideration has a negative impact due to the alternative. With regard to the three evaluation considerations shown in Figure 5, the "cost of methodology" would have the objective of minimizing costs.

2.3.1.2 Specifying Evaluation Measures and Scales

The next step in developing the decision analysis tool is to specify evaluation measures—the scale for measuring the degree of attainment—for each evaluation consideration. Evaluation measures provide a quantifiable value that is used to compare the degree of attainment between alternatives for a particular evaluation consideration. Developing accurate evaluation measures is also an important step in this process as they provide the basis for the decision. Evaluation measures must accurately and precisely capture how well the alternative achieves the consideration or they will not provide a valid comparison between the alternatives.

The next challenge is to develop scales that properly quantify the evaluation measure. A combination of scales may be developed. Natural, direct scales should be used when possible. Scales are considered *natural* when the measurement is widely accepted by the general audience. A *direct* scale is one in which the measurement precisely reflects the degree of attainment. As an example, "expenses in dollars" naturally and directly measure the attainment of an evaluation consideration, such as the "cost of acquiring software development tools." When this is not possible, however, *constructed scales*, or a scale that is developed for a particular problem, can be used. Through these techniques, different aspects of the problem can be addressed in the most natural way, allowing for well-founded decisions. In order to support more controversial scales, evidence from research as well as from the survey should be provided.

Figure 6 is an example of an evaluation measure worksheet that corresponds to the evaluation consideration, "cost of acquiring tools," in the value hierarchy shown in Figure 5. The worksheet provides

a list of questions, or focus points, that the decision maker uses to develop a rating, or score, for the evaluation consideration. For successful application of the worksheets, each alternative being considered must be evaluated on the same focus points.

EVALUATION CONSIDERATION: Cost of Tool Support **OBJECTIVE:** Minimize cost	
Cost Factor	**Cost**
The cost of the software development tool(s)	
The cost of maintenance on the product for the expected duration of use	
The cost to train personnel to use the tool(s)	
The cost to install the tool(s)	
The cost of additional hardware required by the specific tool(s)	
The cost of additional software required by the specific tool(s)	
TOTAL	

Figure 6: Example of an Evaluation Measure Worksheet

2.3.1.3 Building the Multiobjective Value Function

Each evaluation measure scale is a single dimensional value function for the particular aspect of the decision. Building a multiobjective value function from each of these evaluation measures is the next step in building a decision analysis tool. The multiobjective value function is a combination of the single dimensional value functions. It is an additive function that applies weights to each of the single dimensional value functions. As the tool is meant to be applicable to a wide spectrum of problems, specifying the weights in advance with no knowledge of the problem domain or the users' requirements of the system would be premature. However, by setting the weights for each individual problem, the general tool becomes more accurate for the specific problem. Determining the weights of each evaluation consideration is a decision that the software engineer must make based on the problem at hand. Below is a technique for determining weights [18].

1. Consider the evaluation considerations and place them in order of successively increasing value increments.

2. Quantitatively scale each of these value increments as a multiple of the next least important evaluation consideration. Evaluation considerations that have the same level of

importance are given value increments of 1. Otherwise, the value increment assigned should be a number greater than 1.

3. Solve for the smallest value increment by setting the total of all the increments to 1.

4. Use the results of Step 3 to determine the weights for all the evaluation measures.

The benefit of setting the value increment in step 2 to equal 1 for any number of evaluation considerations that have been determined to be of an equivalent level of importance establishes tiers of importance. This allows the decision maker to classify a group of evaluation considerations together for reasons such as criticality to the decision. The decision maker is able to make every consideration its own tier, which is the most general application of this procedure.

As an empirical example, suppose there are three evaluation measures—*Cost of Acquiring Methodology*, *Cost of Tools* and *Component Reusability*. Rating the evaluation measures based on the relative importance the decision maker determines the order of importance to be, in increasing value, *Component Reusability*, *Cost of Acquiring Methodology* and *Cost of Tools*. Now suppose that the software engineer determines the importance *Cost of Acquiring Methodology* is 1.5 times the importance of *Cost of Tools*. Also, *Component Reusability* is 1.25 times more important than *Cost of Acquiring Methodology*. That is,

$$w_{Cost\ of\ Acquiring\ Methodology} = 1.5 \times w_{Cost\ of\ Tools}$$

$$w_{Component\ Reusability} = 1.25 \times w_{Cost\ of\ Acquiring\ Methodology} = 1.25 \times 1.5 w_{Cost\ of\ Tools}$$

Setting the sum of the weights to 1,

$$\begin{aligned} 1 &= w_{Cost\ of\ Acquiring\ Methodology} + w_{Cost\ of\ Tools} + w_{Component\ Reusability} \\ &= (1.5 \times w_{Cost\ of\ Tools}) + w_{Cost\ of\ Tools} + (1.25 \times (1.5 \times w_{Cost\ of\ Tools})) \\ &= w_{Cost\ of\ Tools}\ (1 + 1.5 + (1.25 \times 1.5)) \end{aligned}$$

Hence,

$$w_{Cost\ of\ Tools} = 1/(1 + 1.5 + (1.25 \times 1.5)) = 0.23$$

$$w_{Cost\ of\ Acquiring\ Methodology} = 1.5 \times 0.23 = 0.35$$

$$w_{Component\ Reusability} = 1.25 ? 1.5 ? 0.23 = 0.43$$

Due to round off error, the weights in this case do not equal 1, so a minor adjustment to w_2 is made. This example gives the following weights $w_{Cost\ of\ Tools} = 0.22$, $w_{Cost\ of\ Acquiring\ Methodology} = 0.35$ and $w_{Component\ Reusability} = 0.43$.

2.3.1.4 Developing a Software Analysis Tool

The final step in developing the decision analysis tool is to leverage the computational power of a computer. By developing a software implementation, the user will be spared the agony of solving the single dimensional value functions and the multiobjective value function. Additionally, analysis techniques can be applied by varying the weights used in the multiobjective value function. The software analysis will allow for the user to see graphically the impact the different factors will have on the problem based on the selected weight. The software implementation is a combination of a Java application and a Microsoft Excel workbook. An example of the tool's interface, the criteria selection dialog, is shown in Figure 7.

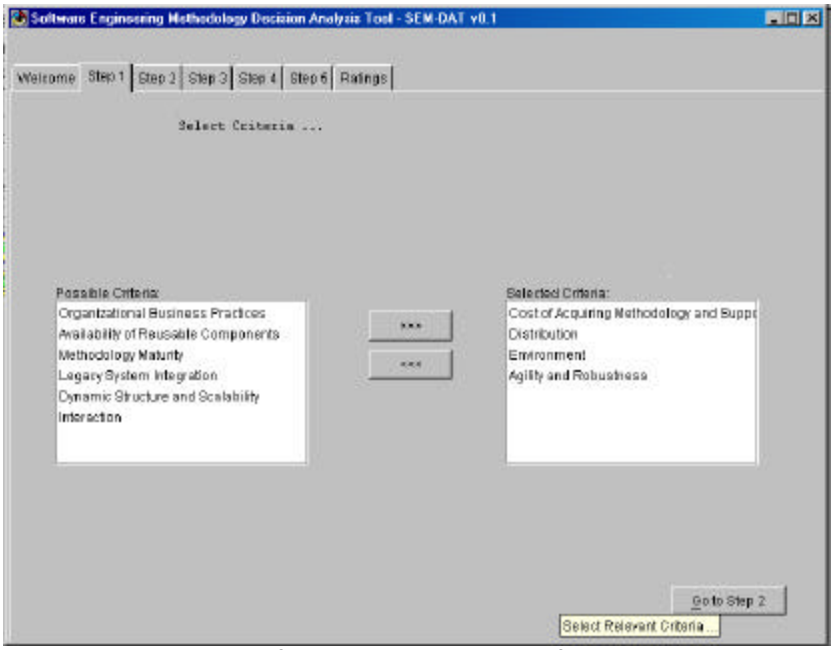


Figure 7: Software Analysis Tool Interface Example

2.3.2 Application of Decision Analysis Tool

The tool provides a fitness value for each of the alternatives that can be used to compare software engineering approaches. The application of the decision analysis tool is based on the software engineer completing the evaluation measure worksheets for each evaluation considerations relevant to the requirements specification. Additionally, the software engineer must determine the weights of each of the evaluation considerations, which gives the decision maker the ability to customize the tool to the problem.

2.3.3 Result Analysis

Guidelines are available to assist the decision-maker based on particular characteristics of the problem domain, but the weights are really assumptions set by the decision-maker in order to capture the relative importance of each evaluation consideration for the particular problem. Techniques, such as sensitivity analysis, provide the decision-maker with the ability to test the weighting assumptions by varying the values of the weights.

The sensitivity analysis provides the decision-maker with the ability to visualize the variability in the decision made by the tool if the weights were assigned different values. Another area that needs to be analyzed pertains to how well the methodologies are represented by the evaluation considerations. In order to validate the decision the tool provides, one of the case studies described in Chapter IV was implemented using the two highest-rated methodologies. Two sets of data were collected on the implementations. First, metrics were collected during each phase of the engineering process. The metrics selected for data collection focus on the productivity of the software engineer [1]. The metrics selected with a description of the measurement are shown in Figure 8.

Metric	Description
Modeling Effort	Measures the number of labor hours spent within the analysis, design, and implementation phases of the software development
Size	Measures the source lines of code (SLOC), and the number of components or classes developed
Complexity	Measures the cyclomatic complexity of the implementation
Size/Effort Ratio	Composed of Modeling Effort and Size, provides a productivity measurement

Figure 8: Validation Metrics

The second set of data collected on implementation was in the form of a questionnaire, which was presented to a group of students in a graduate software engineering course. The course, "Software Evolution," focuses on the post-production phase of software engineering including topics such as software maintenance, programming understanding, reverse- and re-engineering, and designing systems for maintenance. Via the questionnaire, the students were asked to identify a number of technical issues related to the software requirement. In order to answer the questions, each student received a set of analysis and design models produced by following one of two methodologies. The data collected from the questionnaire was used to validate the methodology's ability to represent the technical issues of the problem.

2.4 Summary

This chapter addresses an approach to determining an appropriate software engineering paradigm for a particular problem. This process consists of three phases. Before beginning, it is vital to understand the problem of selecting a software engineering paradigm. There are a number of approaches that can be taken, where each offers some benefit to the software engineer. Understanding the differences in approaches is vital to determining criteria that can be used to base a decision. This understanding led to the selection of the Multiobjective Decision Analysis technique. The first phase of the approach is to develop a method for making a decision based on decision theory. For this particular approach, multiobjective decision analysis is the heart of the process. By following the steps to building a decision analysis tool, an examination into the criteria to base the decision is made. After establishing this set of criteria, the next step is to determine valid evaluation measures to quantify the alternative. Determining a fitness function made up of the many individual evaluation measures is the next step that provides a way to evaluate alternatives over all objectives. The final step in this portion of the approach is to automate the process through a computerized implementation, which allows for fast variation analysis.

Following development of the decision analysis tool, the second phase in the approach is to apply the tool to software problems. This step is required to validate the tool's performance and is conducted on a set of software problem case studies. The final phase is to analyze the results of the metrics collected on

the development of the system and to provide feedback on the accuracy of the paradigm suggested by the decision analysis tool.

Next, Chapter III discusses the development of the decision analysis tool described above. The specifics of the value hierarchy and evaluation measures are developed as the basis for the decision analysis tool.

III. Design

3.1 Introduction

When faced with making a decision, a decision maker is in a better position to explain and defend the decision by using a strategic decision-making technique. Chapter II presents one framework that may be used for developing a decision analysis tool.

Software developers are faced with a great number of alternative software engineering methodologies. The differences between the methodologies can be vast. Some methods are well detailed specifically describing the activities of the tasks in the methodology, while others provide guidance and still others barely address certain issues [5].

In addition to the level of detail, the methodology must also take into account real-world constraints of the particular software problem. The way methodologies handle these constraints that limit the problem solution space often limits the approaches that can be taken to transform the requirements into an acceptable problem solution. In this chapter, the framework that is described in Chapter II is applied to the problem of selecting an appropriate software engineering methodology by measuring the constraints the methodology places on the solution.

3.2 Developing the Decision Analysis Tool

The challenge of building a decision analysis tool for determining an appropriate software engineering methodology is not in the mechanics. The challenge is capturing the details that are important to the software engineer that must make the decision about what methodology is appropriate. Figure 4 provides an overview of the steps taken to develop the tool. The first step in the process is to define a value hierarchy.

3.2.1 Value Hierarchy

Capturing the values of the decision maker in a value hierarchy is the first, and one of the most difficult steps in developing a decision analysis tool. Based on the three areas for characterizing software

methodologies—technical, management, and usage—two major evaluation considerations were derived. Those evaluation considerations are the "Management Issues" and "Project Requirements" as discussed in Chapter II. These categories were further broken down into more specific evaluation considerations. The initial set of considerations was:

- *Management Issues*
  - *Cost of Acquiring the Methodology* – The costs involved with adopting the methodology for use. Factors that impact this category include the costs incurred by sending personnel to available training, the purchase of reference material, etc.
  - *Cost of Acquiring Support Tools* – The costs incurred by purchasing tools that support the methodology. The tools include CASE tools as well as programming development tools. Additionally, the cost of factors such as additional hardware/software to operate the tools, maintenance costs for the tools, and training, should be included.
  - *Availability of Reusable Components* – The incorporation of previously developed software into a new system reduces the overall design, implementation, and testing phases for software development. This category is used to measure the methodology's ability to incorporate predefined components into the system.
  - *Effects on Organizational Business Practices* – This factor measures the impact the adoption of a methodology will have on the existing business practices of the organization. The business practice includes ideas such as tracking development progress through milestones, reports, and customer interactions.
  - *Compliance with Standards* – An alternatives ability to meet standards, whether local to the organization or outside the organization such as national or international, is the factor measured in this category.
  - *Traceability of Changes* – This category measures the methodology's support to trace changes throughout the development lifecycle.

- *Project Requirements*
    - *Legacy System Integration* – The support for the integration of legacy systems with the new project requirement is measured in this category.
    - *Distribution* – The ability to support the modeling of distributed aspects of the problem is the focus of this category.
    - *Environment* – This category is used to measure the methodology's support of developing software systems for environments that have heterogeneous hardware or software.
    - *Dynamic System Structure* – The methodology's ability to develop software capable of handling the introduction and removal of system components in a manner that is not detrimental to the users of the system is considered in this category.
    - *Interaction* – This category determines the methodology's ability to handle the interaction between system-level components as well as entities outside the system such as human users and other systems.
    - *Scalability* – This category measures the methodology's ability to develop software capable of handling the introduction and removal of system-level resources while minimizing the impact on users.
    - *Agility and Robustness* – The focus of this category is to measure the methodology's ability to create flexible software systems that will be resilient to dynamic changes in the environment.

Though these considerations were developed from a number of literature sources, the compiled list was presented to software engineering professionals in academia, industry, and government through a survey questionnaire on the Internet [24, 36]. In order to increase survey participation, an announcement was distributed to software engineering professionals through electronic mail lists maintained by the Object Management Group (OMG), University of Maryland Agent Web, and the Software Engineering Research Network at the University of Calgary. In addition to these broadcast mailings, announcements requesting participation were placed on related, moderated newsgroups—comp.ai and comp.software-eng. Finally,

requests were sent directly to a number of respected academics, researchers, and industry leaders. A copy of the survey that was on the Internet is in Section B.1 of Appendix B.

The period for response collection was set at three weeks. Over that period, thirty-three valid responses were collected. The responses are available in Section B.2 of Appendix B. The survey began with some basic demographic questions in order to develop a profile of the responders. Of the thirty-three responders, twenty-two people indicated that they were associated with the academic community, three responders were associated with government organizations, and eight were associated with the industrial/commercial sector. As for experience, seventeen indicated 1-5 years of experience in their field. Nine responders categorized themselves as having 5-10 years of experience, and seven responders indicated over 10 years of experience.

The survey also collected the opinions of the responders on the importance of the evaluation consideration that have been proposed for the decision as well as their thoughts on the suggested factors, the relative weighting of the management and technical categories, and additional possible factors. As for the evaluation considerations proposed, the responders were asked to rate the considerations on a scale of zero to four. Additionally, responders could leave considerations "not rated". The results of this portion of the survey are in the stacked bar chart in Figure 9. This chart shows the number of scores—NR, 0, 1, 2, 3, or 4—each factor received.

The set of scores each factor received indicates that the responders believed the technical issues are more important that the management issues. Figure 10 shows the average scores each of the considerations received. Again, based on this view of the data, it is clear that the responders felt more emphasis should be on the technical issues of the problem, rather than the issues related to the management factor of the decision.

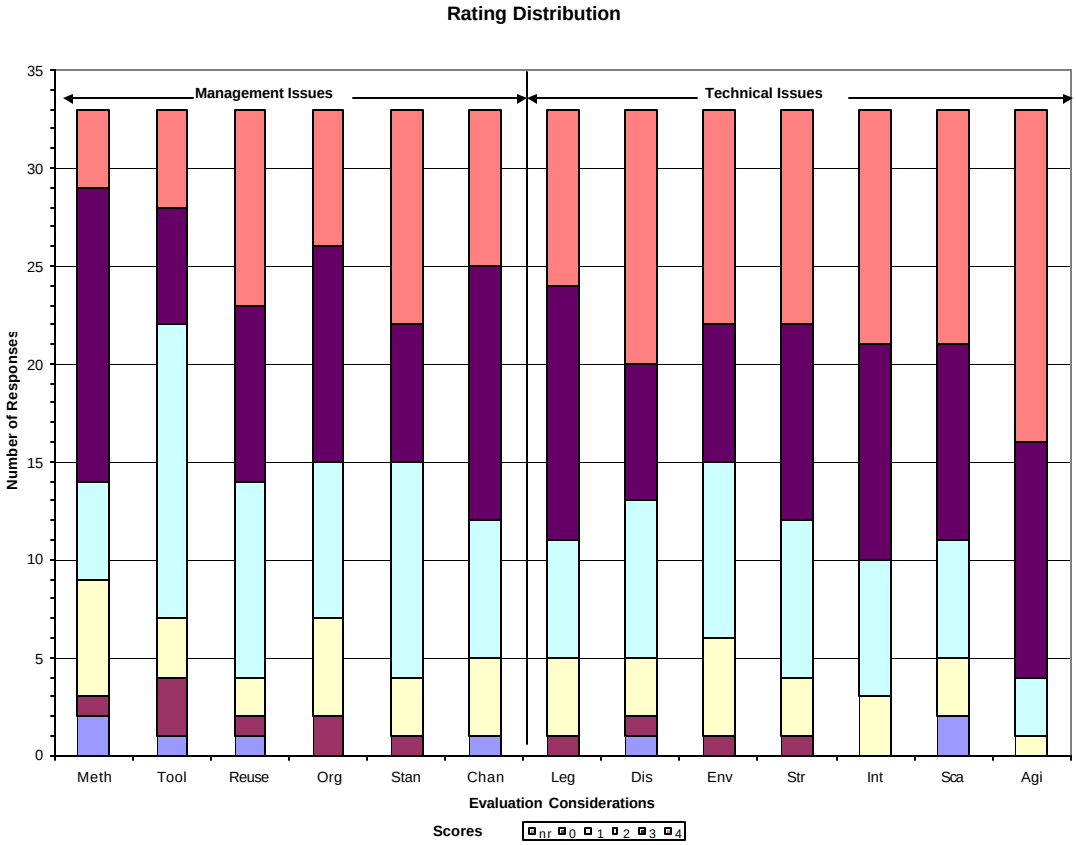


Figure 9: Proposed Evaluation Consideration Ratings

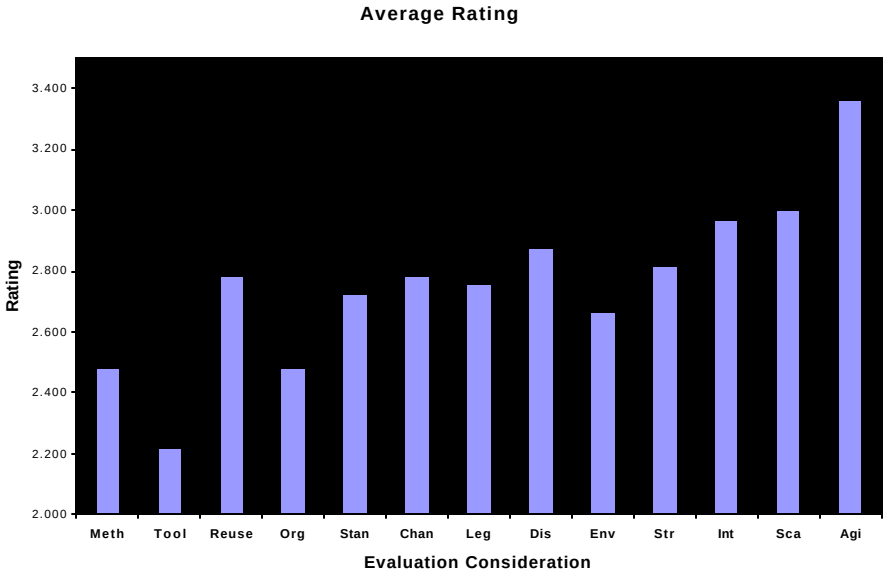


Figure 10: Average Rating for Proposed Evaluation Considerations

The survey also asked whether basing the weights for the evaluation considerations relative to only the other considerations in the same issues category was more appropriate than determining weights relative to all of the considerations. The majority of responses were to determine weights relative to all of the considerations. Most responders did provide an opinion on the total weight each of the major issues. Like the trend seen in Figure 10, fourteen responders felt that the technical issues should impact the decision more than the management issues. On the other hand, five responders felt that the management issues should weigh more on the decision. Three responders indicated that both sets of issues should have an equal weight. The remaining responders did not specify a particular partitioning. Figure 9 shows the data gathered from this particular question.

Management Issues	Technical Issues	Number of Responses
10	90	2
25	75	1
30	70	2
33	66	1
35	65	3
40	60	3
45	55	2
50	50	3
60	40	2
75	25	3
No Partition		11

Figure 11: Weighting Partition

Finally, the survey posed the question: what important factors are missing? Several alternatives were suggested for the cost category. Responders indicated that other factors would have more significance to the problem such as a cost/benefit ratio, cost savings, and productivity gains, because the benefit of the new methodology, if it were great enough, would mitigate any impact that the initial cost would have. Other management factors suggested—availability of tools and experience base—would be appropriate to evaluate the maturity of the methodology. Considerations in this area included the availability of tools as opposed to just the cost, and the experience base of the methodology. Though requested, no suggestions for technical issues were submitted.

Based on the research and the survey results, several changes were made to the list of proposed evaluation considerations. Similar categories, like Dynamic Structure and Scalability, were combined to form a single category, as were Organizational Practices, Compliance with Standards, and Effects of Change; and the Cost of Acquiring the Methodology and Tools. Methodology Maturity was added to the list in order to capture that aspect of the decision. The complete value hierarchy is shown in Figure 12. In addition to specifying the evaluation considerations, other factors such as objectives, goals and evaluation measures are required to fully define the values of the decision maker. The next section looks at these other factors while further developing the evaluation measures that are used to evaluate the different alternatives.

3.2.2 Evaluation Considerations, Measures and Objectives

This section looks at the further decomposition of the two major evaluation considerations, as well as describing the other factors that are important in the decision making process, i.e., the objectives of each of the categories and the definition of the evaluation measures that is used to rate each alternative with respect to the consideration. For each evaluation consideration, a set of related measurement questions, or focus points, is provided to ascertain the degree of attainment the methodology has for the particular consideration.

By using the focus points to measure the level of attainment the methodology achieves for a given consideration, it is necessary to use a constructed scale for rating the alternatives. In other selection type problems, such as the methodology selection for real-time systems (see Section A.5.3) the focus points are answered simply yes or no [38]. Unfortunately, the focus points tend to be subjective. As such, it is difficult to analyze and compare methodologies at this level of granularity. In order to derive a more refined rating, the evaluation considerations are measured using a zero-to-four rating system with one exception.

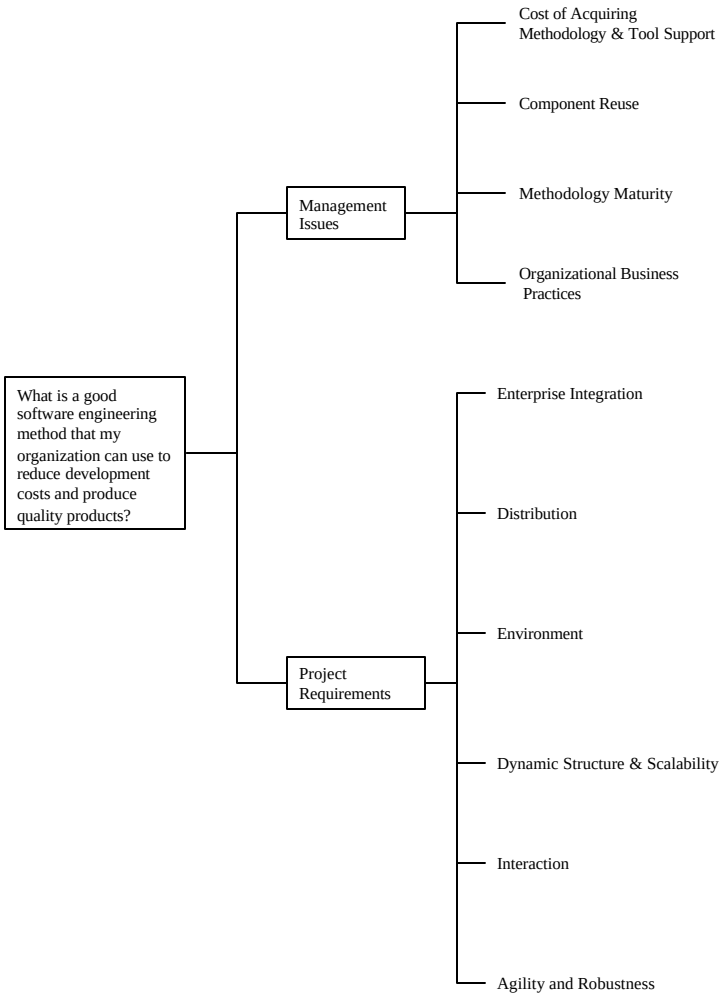


Figure 12: Methodology Selection Value Hierarchy

The zero-to-four rating system is a five-point scale used for each focus point. Compared to the methodology selection for developing real-time systems approach, a score of zero represents a "no" and a score of four represents a "yes." This scale also allows three intermediate levels of achievement, giving the decision-maker the ability to credit methodologies that recognize the issue and provide some level of

support for the issue. As this technique does nothing to reduce the level of subjectivity on which the alternative's rating is based, the decision-maker must be careful to use the same level of scrutiny for each alternative analyzed. In order to assist the decision-maker and to attempt to standardize ratings, guidance for rating each focus point is included in the discussion of the evaluation considerations.

3.2.2.1 Management Issues

The first category of evaluation considerations is based on the management and usage characterizations of software methodologies. Using the phrase "management issues" to cover these characteristics, the evaluation considerations under this branch of the value tree focus on the issues related to using a methodology to develop software.

One evaluation consideration of the management issues sub-tree, the *Cost of Acquiring Methodology and Support Tools*, focuses on the costs associated with selecting a new methodology. Costs that an organization will be faced with include acquiring software tools for project development, reference material and personnel training. The impact of the cost of a methodology could be a significant factor in the decision, particularly if the organization is made up of "experts" with regard to one alternative and "novices" with regard to another.

Another evaluation consideration of the management issues sub-tree, *Organizational Business Practices*, is the impact the methodology will have on the current business practices of the organization. An organization is in a much better position to make estimates regarding software projects based on the experience of previous software projects [31]. Following a well-defined process is also an underlying idea of improvement processes such as the Capability Maturity Model (CMM). For an organization that is meeting CMM requirements, management would likely prefer a methodology that would allow the organization to continue to meet those requirements.

In the next four sections, the lowest level of evaluation considerations for management issues is explored. Each of the evaluation considerations is discussed with regard to the objective of the

consideration as well as the criteria-based questions that the decision-maker will use to evaluate alternatives.

3.2.2.1.1 Cost of Acquiring Methodology and Support Tools

A software engineering methodology is an investment that an organization makes in order to ensure quality products. The use of the methodology affords the organization a repeatable process on which estimates can be made for future projects. The process, however, depends upon the knowledge and experience of the software professionals to make reasonable assumptions for project estimation. When faced with the decision of selecting a new methodology, the decision-maker must take into account the financial impact to the organization in order to adopt the methodology.

To properly use a methodology, education and training are key aspects that must be explored. Academic institutions approach the instruction of software engineering in many different ways. For the most part, the focus is not on specific methodologies, but rather the underpinnings of several methodologies, which give students a broader exposure to various software engineering techniques [22]. Additionally, experience is a major factor in producing software, as not every problem has a textbook solution. Software engineering is a balance of art and science [31]. Providing the training for the organization's software developers is required to ensure the science, where as practice is required to perfect the art. The two focus points in this category are intended to help the decision maker evaluate the financial impact on the organization if a methodology is selected.

1. *The cost to train personnel to use the methodology* – There are many options available to an organization in order to provide training to its employees. Some managers may decide to send their software professionals to training programs, while others may develop in-house training classes. In either case, there are costs involved. The most obvious cost is that of an actual training class or seminar. There are other factors involved. For example, the organization must fund travel expenses for employees to attend training classes that are not within the local area. Manpower is also lost on current, existing projects when people are sent away to training.

2. *The cost of additional reference material on the methodology* – Sending personnel to training classes is an important step in properly learning the methodology, but another area that the organization needs to consider is additional reference material for the employees. The costs of additional reference material may include textbooks, professional journal/magazine subscriptions, and computer-based training as an example. As regards the computer-based training, it is important to combine cost items in one category, if the computer-based training software is the preferred solution for personnel training, then it should not be included in this rating. If on the other hand, the computer-based training software is supplemental to another training program, it is appropriate to include it in this category.

In addition to acquiring the methodology, the organization incurs costs from the acquisition of software support tools. Software developers gain many benefits from tool support of the methodology. The tool ensures the information required for each representation is collected and the rules of the method are enforced. Without such support, the designer must ensure each representation fulfills the requirements of the models. Additionally, tools can provide automated analysis of models to ensure correctness. Automated tools can also provide transformation support between different models and phases of the methodology [5].

The focus of this evaluation consideration is to compare the cost of purchasing similar tools for the alternative methodologies. Like the previous evaluation consideration, the costs of acquiring tools could be rated in many different ways. Costs can be recorded based on the actual amount at purchase time, or they can be normalized over a "per project" range. Details of the focus points are below.

3. *The cost of the software development tool(s)* – This factor looks at the costs for the tools that directly support the methodology. In order to make a valid comparison, the tools must support the same phases of development. This can be done by using a normalized cost for the support development tool, which can be derived by dividing the cost of the tool by the number of development phases the methodology supports.

4. *The cost of maintenance on the product for the expected duration of use* – After purchasing the software, it may require contracted maintenance. The maintenance contract should be purchased as required, but all alternatives should have a similar plan. In the event that similar options are not available, the decision maker should estimate the potential cost a maintenance event would cost the organization.

5. *The cost to train personnel to use the tool(s)* – Like educating the organization on how to use the methodology, training for how to use the tool should also be considered.

6. *The cost to install the tool(s)* – Installing the tools on the organization's systems is also a factor to consider when determining the cost of the tool. Many options are available to organizations for the distribution of software. The particular option as well as labor costs should be considered.

7. *The cost of additional hardware required by the specific tool(s)* – In the event that the tools require a hardware platform that the organization does not have, the new system would need to be purchased.

8. *The cost of additional software required by the specific tool(s)* – Likewise, a tool may require a specific type of software to be running, such as an operating system, in order to operate. Unless the software is available, the tool is unusable.

When evaluating a set of alternatives, it is important to compare them using the same set of factors. In cases where options for training are available for one methodology but not another, the decision maker needs to determine if a similar option would achieve the same objective or if it is necessary to develop in-house training. For example, a training seminar may be available for one methodology, but not another. An alternative would be to contract with an "expert" in the methodology to develop a course. Another important issue when comparing methodologies in this category is normalization. One way to normalize the costs is to determine a "per person" value. An alternative normalization would be "per

project." Again, in order to make an accurate comparison, the same normalization must be used for all alternatives.

The objective of this evaluation consideration is to minimize the cost of acquiring the methodology and support tools. A summary of the factors is in the evaluation consideration worksheet in Figure 13. When completing the worksheet, the decision maker should use the normalized cost for each focus point.

EVALUATION CONSIDERATION: Cost of Acquiring Methodology and Support Tools **OBJECTIVE:** Minimize cost	
Cost Factor	**Cost**
The cost to train personnel to use the methodology	
The cost of additional reference material on the methodology	
The cost of the software development tool(s)	
The cost of maintenance on the product for the expected duration of use	
The cost to train personnel to use the tool(s)	
The cost to install the tool(s)	
The cost of additional hardware required by the specific tool(s)	
The cost of additional software required by the specific tool(s)	
TOTAL	

Figure 13: Cost of Acquiring Methodology and Support Tools Worksheet

3.2.2.1.2 Organizational Business Practices

Developing software, like any product, requires more than just the methodology for software engineering. Most organizations develop "in-house" standards and practices to which the software projects adhere. Additional activities, such as planning, organizing and staffing, and project tracking and control, are developed to manage costs and personnel [38]. These activities usually are developed from lessons learned during past projects, so management is committed to the activities as they corrected the problems encountered during previous project developments.

Management processes are usually time driven, as they are developed to estimate project time for potential projects. As such, resources like manpower and money can effect the time on the project. Software development is generally event driven, where one phase begins after another ends [38]. With that in mind, management establishes project milestones for developers to complete. Tracking the milestones and deliverables allows for monitoring the progress of the development. The method should be able to

work within the project management framework. The following focus points are used to evaluate alternatives with respect to the organization's process.

1. *The method provides planning techniques that lead to milestone definitions and project plans that are consistent with use of the method and the implementation language* – This focus point is used to measure the method's ability to lend its models to the existing project management framework. It will probably be necessary for both the project management framework and methodology to be flexible, i.e. if the project management framework required five milestones based on five phases of a methodology and an alternative methodology only has three phases, either certain milestones can be removed or the methodology's phases could be broken down into five steps. Additionally, the method should be judged to fit within the current management process with major modifications to models.

Guidance:

4 - Methodology defines a macro- and micro-level development process compatible with existing project management framework

2 - Methodology describes a micro-level development process that can be incorporated with the organizations macro-level project management framework

0 - Methodology describes a set of models with no process

2. *The method has analysis techniques that can be used during reviews or that can help you gauge progress during development* – The ability to perform analyses on the project during the development can provide the ability to ensure consistency and completeness. Catching problems early in the analysis and design are much less expensive to fix than during the implementation and post-development phases.

Guidance:

4 - Models are representational of the progress being made during development

2 - Models are indirectly representational of the progress being made (for example, by model versioning)

0 - Models are not representative of development progress

3. *Representations are clear and easy enough to understand to be used for design reviews* – A variety of techniques are available for representing the models of the methodology. Some representations, such as a graphical representation, are easier to understand by a larger audience than others, like formal methods [8]. This focus point looks at the ability of using the representations directly from the method in order to describe the characteristics of the proposed system to other interested parties.

Guidance:

4 - Models are familiar and easy to understand

2 - Models are complex, but understandable

0 - Models are unfamiliar and complex

4. *Ability to rapidly develop high-level design representations that can be analyzed to determine the most feasible design approach* – It is beneficial to be able to prove the feasibility of certain requirements prior to implementing the entire solution. Likewise, evaluating different design alternatives at a high-level can ensure the best approach is taken.

Guidance:

4 - High-level designs can be rapidly developed for analysis

2 - High-level designs can be rapidly developed, but analysis techniques are not defined

0 - Rapid development is not available

5. *The method helps partition the system into manageable pieces that can be given to development team members* – The ability to clearly delineate the phases of the methodology allows for partitioning the project among different members of the development team.

Guidance:

4 - The set of models representing an entity will be independent of each other

2 - The set of models will have dependencies, but subsets of the models can be partitioned

0 - The dependency of the models is high, making partitioning difficult

Unless the organization is new, it is likely that there is some software development process in place. As such, when faced with the decision to use a particular methodology, the methodology needs to be able to be incorporated into that process. The representations of the methodology should be able to comply with the applicable standards established by the process. Here is a focus point to consider when evaluating alternatives based on their ability to comply with standards:

6. *The method establishes and enforces well-defined representation standards for each view at each stage of development* – By establishing and enforcing "well-defined" representations, the methodology is providing a degree of formality. The greater the degree of formality that the representation has, the less obscure and open to interpretation the representation is. Furthermore, having standards set for the representations allows for improved understanding of a project across the organization [5].

Guidance:

4 - Syntax and semantics of the models are well-defined

2 - Syntax of models is well-defined

0 - Syntax and semantics are not well-defined

As projects progress through the phases of a software engineering methodology, it is inevitable that changes will be made either by a requirement change or a clarification from the customer. The methods ability to handle the changes throughout the development process should be inherent, allowing the designer to refine and enhance the design as well as evaluate design alternatives [5]. The following focus point is provided for evaluating the methods capability for tracing changes.

7. *The method provides hierarchical forms for all types of representations* – With a hierarchical form of representation, it is important that if a change is made, it is consistent throughout the models. This factor measures the method's ability to allow changes at lower levels without impacting the "big picture" with regard to the higher levels. Likewise, this factor measures the method's ability to allow changes at the higher levels without impacting the lower levels.

Guidance:

4 - Low-level and high-level changes can be made without effecting representations at other levels

2 - Low-level changes can be made without effecting the high-level representations

0 - Changes cannot be isolated

Using the zero-to-four rating scale, Figure 14 summarizes the focus points for this evaluation consideration.

EVALUATION CONSIDERATION: Organizational Business Practices **OBJECTIVE:** Maximize Rating	
FACTOR	**RATING**
The method provides planning techniques that lead to milestone definitions and project plans that are consistent with use of the method and the implementation language	[0 1 2 3 4]
4 - Methodology defines a macro- and micro-level development process compatible with existing project management framework 2 - Methodology describes a micro-level development process that can be incorporated with the organizations macro-level project management framework 0 - Methodology describes a set of models with no process	
The method has analysis techniques that can be used during reviews or that can help you gauge progress during development	[0 1 2 3 4]
4 - Models are representational of the progress being made during development 2 - Models are indirectly representational of the progress being made (for example, by model versioning) 0 - Models are not representative of development progress	
Representations are clear and easy enough to understand to be used for design reviews	[0 1 2 3 4]
4 - Models are familiar and easy to understand 2 - Models are complex, but understandable 0 - Models are unfamiliar and complex	
Ability to rapidly develop high-level design representations that can be analyzed to determine the most feasible design approach	[0 1 2 3 4]
4 - High-level designs can be rapidly developed for analysis 2 - High-level designs can be rapidly developed, but analysis techniques are not defined 0 - Rapid development is not available	
The method helps partition the system into manageable pieces that can be given to development team members	[0 1 2 3 4]
4 - The set of models representing an entity will be independent of each other 2 - The set of models will have dependencies, but subsets of the models can be partitioned 0 - The dependency of the models is high, making partitioning difficult	
The method establishes and enforces well-defined representation standards for each view at each stage of development	[0 1 2 3 4]
4 - Syntax and semantics of the models are well-defined 2 - Syntax of models is well-defined 0 - Syntax and semantics are not well-defined	
The method provides hierarchical forms for all types of representations	[0 1 2 3 4]
4 - Low-level and high-level changes can be made without effecting representations at other levels 2 - Low-level changes can be made without effecting the high-level representations 0 - Changes cannot be isolated	
TOTAL	/28

Figure 14: Organizational Business Practices Worksheet

3.2.2.1.3 Methodology Maturity

The maturity of the methodology is a factor that can play an important part in the decision process. Methodologies become more mature as more software developers make use of them. In order to measure the maturity of the methodology, the focus points attempt to derive the availability of support tools and of published training and reference material. In addition to the availability of tools and reference material, the number and type of systems that have been created with the paradigm indicate maturity of the methodology. The fact that systems have been developed using the methodology provides some level of validation that the methodology is successful. The following focus points are provided for evaluating methodology alternatives with regard to methodology maturity:

1. *Support tools are available for the methodology* – The availability of tools may vary greatly. For those methodologies that are entrenched in mainstream industry, tools can span the gambit of low-cost educational aids to high-cost industrial strength tools. The investment that companies make within these tools indicates the acceptance of the methodology. On the other side of the spectrum, new methodologies may have no tools designed to support them, or those that do may be experimental, or developed to demonstrate concepts. Ratings in this category should indicate higher scores for methodologies with "industrial strength" tool support, and lower scores for methodologies with "experimental" tool support.

   Guidance:

   4 - Multiple commercially developed tools are available

   2 - Multiple academically developed research tools are available

   0 - Support tools are not available

2. *Training/reference material is available for the methodology* – A review of available literature on the methodology is also helpful for determining the maturity of the methodology. The rating system in this case should score a methodology that has been published as a book,

higher than one that is published in journals or conference papers. There should also be some discrimination against the type of publication when considering the methodologies.

Guidance:

4 - Published textbooks and training manuals, commercially developed training seminars, or computer based training are available

2 - Reference material has been published in journals and conference proceedings

0 - Reference material may be limited to technical reports and white papers, and training is not available

3. *Systems developed with the methodology exist* – The last focus point in this consideration looks at the history of the methodologies use. A methodology that has been employed to create industrial strength applications should be given a higher score than one that has only been used to develop small demonstration projects.

Guidance:

4 - > 500 Systems developed

2 - 50 – 200 Systems developed

0 - < 5 Systems developed

Figure 15 summarizes the methodology maturity evaluation consideration, and uses the zero-to-four rating system for scoring alternatives.

EVALUATION CONSIDERATION: Methodology Maturity **OBJECTIVE:** Maximize Rating	
FACTOR	**RATING**
Support tools are available for the methodology	[0 1 2 3 4]
4 - Multiple commercially developed tools are available 2 - Multiple academically developed research tools are available 0 - Support tools are not available	
Training/reference material is available for the methodology	[0 1 2 3 4]
4 - Published textbooks and training manuals, commercially developed training seminars, or computer based training are available 2 - Reference material has been published in journals and conference proceedings 0 - Reference material may be limited to technical reports and white papers, and training is not available	
Systems developed with the methodology exist	[0 1 2 3 4]
4 - > 500 Systems developed 2 - 50 – 200 Systems developed 0 - < 5 Systems developed	
TOTAL	/12

Figure 15: Methodology Maturity Worksheet

3.2.2.1.4 Integration of Reusable Components

By using existing components, the development of the system should be reduced as the component is known to work appropriately and the time involved with tying the component into the system would be less than the time it would take to "redevelop" the component [5]. Since it is desirable to reuse components, they should only be reused when they assist in the development of the system. Component reuse should not be considered too early in the development process, as the system developed may not exhibit the operational characteristics desired by the customer. Below are the focus points to consider when evaluating the methodology's support of component reuse.

1. *The method encourages separating requirements into essential and negotiable classes, where negotiable classes identify the opportunity for reuse* – Not all components designed for reuse meet the same operational characteristics. If a requirement is essential, it may not be possible to find a component guaranteed to meet those operational characteristics, in that case, reuse is not possible.

The negotiable classes, on the other hand, do not have the strict requirement, so, there is room to evaluate the tradeoffs between different components.

Guidance:

4 - Models differentiate between essential and negotiable classes and representations are available for previously developed components

2 - Representations are available to incorporate components developed previously by this methodology

0 - Essential and negotiable classes are not differentiated and representations for previously developed components do not exist

2. *The method encourages designers to suggest alternatives provided by existing components to negotiable requirements* – The method's support of component reuse is more valuable if it allows the evaluation of alternative design choices.

Guidance:

4 - Methodology suggests a spiral approach in which alternatives can be integrated with system for evaluation

2 - Methodology suggests evaluation of alternatives, but decision should be made before proceeding

0 - Methodology suggests no evaluation of design alternatives

As software reuse is a goal of most software engineering methodologies, it is important to look at how well the alternative methods support reuse. The focus points are summarized in the worksheet shown in Figure 16.

EVALUATION CONSIDERATION: Reusable Components **OBJECTIVE:** Maximize Rating	
FACTOR	**RATING**
The method encourages separating requirements into essential and negotiable classes, where negotiable classes identify the opportunity for reuse	[0 1 2 3 4]
4 - Models differentiate between essential and negotiable classes and representations are available for previously developed components 2 - Representations are available to incorporate components developed previously by this methodology 0 - Essential and negotiable classes are not differentiated and representations for previously developed components do not exist	
The method encourages designers to suggest alternatives provided by existing components to negotiable requirements	[0 1 2 3 4]
4 - Methodology suggests a spiral approach in which alternatives can be integrated with system for evaluation 2 - Methodology suggests evaluation of alternatives, but decision should be made before proceeding 0 - Methodology suggests no evaluation of design alternatives	
TOTAL	/8

Figure 16: Reusable Components Worksheet

3.2.2.2 Project Requirements

The second category of evaluation considerations is based on the technical characterizations of software methodologies. Using the phrase "project requirements" to cover these characteristics, the evaluation considerations under this branch of the value tree focus on the issues related to functional, behavioral and structural aspects of the project to be developed.

The focus of these technical characteristics is derived from current literature with regard to the issues that are important to the next generation of business-oriented information systems [36]. This domain has many characteristics that are similar to other domains, particularly with regard to heterogeneous environments, system integration, and geographic distribution of resources. Identifying the strengths and weaknesses of the methodologies with regard to how well the method supports these considerations is key to developing the system in a high quality, low cost manner.

3.2.2.2.1 Legacy System Integration

One of the major benefits of software is the ability to automate processes that are tedious and repetitious to humans. At this point in time, many processes are automated, yet many are not due to the complexity of the process or the inability of the organization to fund automation projects. Because of these reasons, there tends to be a piecemeal development of projects to automate the organization's processes. When new projects arise, the ability to tie the product of the new project to existing software products has emerged as a goal to be considered [36]. The purpose of the focus points for this evaluation consideration is to rate the methodologies ability to tie the new system into the organization's related systems.

1. *Methodology provides representations to allow the incorporation of a legacy system if it has a documented interface* – Based on a documented interface for the legacy system, the methodology should provide some form of representation to model it in the new system. This is important in order to ensure communication between the systems is correct, as well as to ensure compatibility issues with data types, etc., are handled properly.

Guidance:

4 - Representations model the legacy system from a structural, functional, and behavioral aspect

2 - Representations capture the structural view of the legacy system

0 - Representations to model legacy systems are not available

2. *Methodology provides a representation for referencing legacy system functions/methods* – If the legacy system is directly compatible with the proposed system, this focus point evaluates how well the methodology supports the incorporation of the legacy systems functionality.

Guidance:

4 - Legacy system is modeled as a white box system, allowing the integration of interfacing methods as well as reuse of intermediate methods

2 - Legacy system is modeled as a black box system, allowing the integration of interfacing methods

0 - Representations to model legacy system's functions or methods are not available

The factors of this category are rated on the zero-to-four scale. For this evaluation consideration, the objective is to maximize the rating. A summary of the focus points is in the worksheet in Figure 17.

EVALUATION CONSIDERATION: Legacy System Integration **OBJECTIVE:** Maximize Rating	
FACTOR	**RATING**
Methodology provides representations to allow the incorporation of a legacy system if it has a documented interface	[0 1 2 3 4]
4 - Representations model the legacy system from a structural, functional, and behavioral aspect 2 - Representations capture the structural view of the legacy system 0 - Representations to model legacy systems are not available	
Methodology provides a representation for referencing legacy system functions/methods	[0 1 2 3 4]
4 - Legacy system is modeled as a white box system, allowing the integration of interfacing methods as well as reuse of intermediate methods 2 - Legacy system is modeled as a black box system, allowing the integration of interfacing methods 0 - Representations to model legacy system's functions or methods are not available	
TOTAL	/8

Figure 17: Legacy System Integration Worksheet

3.2.2.2.2 Distribution

Another fundamental requirement for complex software systems is distribution [36]. The exact form of the distribution in the problem can vary. First, the requirement may require integrating information or resources that are physically distributed among several systems [23, 38]. A different type of distribution may come from a decision to distribute computation among several systems [15, 38]. In either case, the ability to satisfy the requirement is aided by a methodology that provides representational support of distributed elements, as it assists the developer in making decisions about the appropriate implementation of the software system. For this evaluation consideration, the following focus points are provided.

1. *The method provides a representation for specifying system network configuration* – This focus point assists the decision maker in evaluating the methodology's ability to present the potential impact of different system configurations. When rating this category, it should not be simply a case of whether or not the representation is there, but how much detail is captured within the representation.

Guidance:

4 - Advanced models are available that capture data such as processor load, network latency, etc.

2 - Models are available that capture essential network configuration data, processor type, nodes, addresses, etc.

0 - Network configuration models are not available

2. *The method provides a representation for specifying deployment of entities throughout the network* – This focus point expands on the previous with regard to detail that is inherent to the representation. Specifically, the ability to annotate the representation with the deployment of software components is measured here.

Guidance:

4 - Models visually represent specific entities on network components

2 - Models correlate entities to network components

0 - Representations not available

3. *The method provides the ability to specify locations of resources throughout the system network* – Again, this focus point looks at a specific detail that is in the representation of the system configuration: resources.

Guidance:

4 - Models visually represent specific resources on network components

2 - Models correlate resources to network components

0 - Representations not available

The factors of this category are rated on the zero-to-four scale. For this evaluation consideration, the objective is to maximize the rating. A summary of the focus points is in the worksheet in Figure 18.

3.2.2.2.3 Environment

The next requirement for complex systems is the ability to work within a heterogeneous hardware and software environment. This evaluation consideration attempts to measure the methodology's support for implementing system designs for multiple platforms. The ideal, of course, is to be able to have a single implementation that works on all required platforms, but this is not always possible. The ability to minimize the amount of rework needed to port the code to available platforms is important as it reduces the opportunity for the introduction of bugs [31]. Below are the proposed focus points for evaluating the methodology's support of heterogeneous environments.

EVALUATION CONSIDERATION: Distribution **OBJECTIVE:** Maximize Rating	
FACTOR	**RATING**
The method provides a representation for specifying system network configuration	[0 1 2 3 4]
4 - Advanced models are available that capture data such as processor load, network latency, etc. 2 - Models are available that capture essential network configuration data, processor type, nodes, addresses, etc. 0 - Network configuration models are not available	
The method provides a representation for specifying deployment of entities throughout the network	[0 1 2 3 4]
4 - Models visually represent specific entities on network components 2 - Models correlate entities to network components 0 - Representations not available	
The method provides the ability to specify locations of resources throughout the system network	[0 1 2 3 4]
4 - Models visually represent specific resources on network components 2 - Models correlate resources to network components 0 - Representations not available	
TOTAL	/12

Figure 18: Distribution Worksheet

1. *The programming language for implementation is supported by the software/hardware systems on which the program will be deployed* – In general, a methodology is developed to work theoretically, without a specific implementation language in mind. The representations in the methodology may, however, translate more readily into a particular paradigm of programming languages, i.e. object-oriented methodologies are realized more appropriately in an object-oriented language like C++ rather than a procedural language such as Pascal.

Guidance:

4 - Language is supported by software/hardware systems requiring no specialized compilers/interpreters/translators

2 - Language is supported by software/hardware systems, but special compilers/interpreters/emulators are required for particular systems

0 - Language is not supported by software/hardware system

2. *The methodology supports deployment modeling in which the engineer can easily determine necessary changes in the case of system specific commands* – When faced with developing systems for heterogeneous environments, the ability to specify details of the system configuration in deployment diagrams provides developers with vital information when considering issues such as system specific commands, such as system calls to the operating system.

Guidance:

4 - Hardware and software environment and entities are visually mapped onto system deployment diagrams

2 - Correlation between entities and systems exist

0 - System deployment is not modeled

3. *The programming languages supported by the methodology for implementation requires little additional time for compilation for each system type* – Developing one version of the implementation that can be used by all systems makes software maintenance less troublesome.

Guidance:

4 - Programming languages do not require specialized libraries for particular systems

2 - Some sub-components require specialized libraries (for example graphic components) for particular systems

0 - All components require specialized libraries

The factors of this category are rated on the zero-to-four scale. For this evaluation consideration, the objective is to maximize the rating. A summary of the focus points is in the worksheet in Figure 19.

EVALUATION CONSIDERATION: Environment **OBJECTIVE:** Maximize Rating	
FACTOR	**RATING**
The programming language for implementation is supported by the software/hardware systems on which the program will be deployed	[0 1 2 3 4]
4 - Language is supported by software/hardware systems requiring no specialized compilers/interpreters/translators 2 - Language is supported by software/hardware systems, but special compilers/interpreters/emulators are required for particular systems 0 - Language is not supported by software/hardware system	
The methodology supports deployment modeling in which the engineer can easily determine necessary changes in the case of system specific commands	[0 1 2 3 4]
4 - Hardware and software environment and entities are visually mapped onto system deployment diagrams 2 - Correlation between entities and systems exist 0 - System deployment is not modeled	
The programming languages supported by the methodology for implementation requires little additional time for compilation for each system type	[0 1 2 3 4]
4 - Programming languages do not require specialized libraries for particular systems 2 - Some sub-components require specialized libraries (for example graphic components) for particular systems 0 - All components require specialized libraries	
TOTAL	/12

Figure 19: Environment Worksheet

3.2.2.2.4 Agility and Robustness

Another technical requirement of complex systems is fault tolerance, or robustness [36]. Most project requirements require some level of robustness [38]. In fact, robustness considerations may dictate the system architecture. To some degree the system should be fault tolerant at the system and sub-system level so as to detect and recover from failures at any level, as well as minimize the impacts of the errors on the working environment [36]. The aim of the focus points for this consideration is to measure the methodology's support of developing agile and robust software.

1. *The methodology's representations provide the ability to model normal processing and exception processing* – This focus point considers the methodology's ability to partition normal processing events and exceptional events. By considering each type appropriately, the system can be developed to operate correctly under both types of situations.

Guidance:

4 - Representations are available allowing normal and exceptional processing to be differentiated

2 - Representations are available to model processing

0 - Representations not available

2. *The method provides techniques to model dynamic system reconfiguration* – The ability for a system to continue providing services when certain system resources are added or deleted is an important factor in robustness. This focus point examines whether or not the methodology considers these issues.

Guidance:

4 - Behavior models show the instantiation of components based on trigger events or non-triggered (pseudo random) events

2 - Behavior models (such as sequence diagrams) show the instantiation of components based on trigger events

0 - Representations not available

3. *The method provides techniques to analyze system performance for all configurations* – Analyzing and testing possible configurations of the system is an important area as it can prevent problems that may not otherwise be found until implementation.

Guidance:

4 - Techniques exist for analyzing static and dynamic configurations

2 - Techniques exist for analyzing statically defined configuration

0 - Technique not available

The factors for agility and robustness are rated using the zero-to-four scale. For this evaluation consideration, the objective is to maximize the rating. A summary of the focus points is in the worksheet in Figure 20.

EVALUATION CONSIDERATION: Agility and Robustness **OBJECTIVE:** Maximize Rating	
FACTOR	**RATING**
The methodology's representations provide the ability to model normal processing and exception processing	[0 1 2 3 4]
4 - Representations are available allowing normal and exceptional processing to be differentiated 2 - Representations are available to model processing 0 - Representations not available	
The method provides techniques to model dynamic system reconfiguration	[0 1 2 3 4]
4 - Behavior models show the instantiation of components based on trigger events or non-triggered (pseudo random) events 2 - Behavior models (such as sequence diagrams) show the instantiation of components based on trigger events 0 - Representations not available	
The method provides techniques to analyze system performance for all configurations	[0 1 2 3 4]
4 - Techniques exist for analyzing static and dynamic configurations 2 - Techniques exist for analyzing statically defined configuration 0 - Technique not available	
TOTAL	/12

Figure 20: Agility and Robustness Worksheet

3.2.2.2.5 Dynamic Structure and Scalability

The next functional requirement of software projects deals with scalability. The focus points in this category are directed at the methodology's support of designing systems that allow the incorporation of additional resources and software components with minimal user disruption [36]. Dynamic integration is beneficial to the user because the system does not "go down." Additionally, an approach that supports this characteristic is forward-looking, as the system may some day be a "legacy system" that needs to integrate with a new system. A system designed with this in mind will have clear interfaces for integration. Below are the focus points to rate this evaluation.

1. *The methodology provides techniques for dynamic integration of new subsystems/resources with minimal disruption to existing system* – The ability to add new subsystems and resources without disturbing normal operation is a factor important to those that are relying on the system. The methodology that accounts for this through dynamic integration techniques reduces the impact of fielding new system features.

Guidance:

4 - Techniques for adding resources/subsystem allow on-line updating

2 - Techniques for adding resources/subsystems require system to be taken off-line

0 - Technique not available

2. *The methodology provides techniques for the removal of existing subsystems/resources with minimal disruption to remaining system* – Just as adding subsystems and resources seamlessly, techniques within the methodology for the removal of these entities are also important, as it allows system operation, at perhaps a degraded state.

Guidance:

4 - Techniques for removing resources/subsystem allow on-line updating

2 - Techniques for removing resources/subsystems require system to be taken off-line

0 - Technique not available

The factors of this category are rated on the zero-to-four scale. For this evaluation consideration, the objective is to maximize the rating. A summary of the focus points is in the worksheet in Figure 21.

EVALUATION CONSIDERATION: Dynamic Structure and Scalability **OBJECTIVE:** Maximize Rating	
FACTOR	**RATING**
The methodology provides techniques for dynamic integration of new subsystems/resources with minimal disruption to existing system	[0 1 2 3 4]
4 - Techniques for adding resources/subsystem allow on-line updating 2 - Techniques for adding resources/subsystems require system to be taken off-line 0 - Technique not available	
The methodology provides techniques for the removal of existing subsystems/resources with minimal disruption to remaining system	[0 1 2 3 4]
4 - Techniques for removing resources/subsystem allow on-line updating 2 - Techniques for removing resources/subsystems require system to be taken off-line 0 - Technique not available	
TOTAL	/8

Figure 21: Dynamic Structure and Scalability Worksheet

3.2.2.2.6 Interaction

The final technical evaluation consideration is interaction. Interaction deals with the products ability to deal with the human user. Human interface is an important piece of any software, and one that is most likely to change in order to satisfy the user [38]. The focus points in this category are directed at the

methodology's ability to deal with human factors as well as it's support of architecturally separating the user interface components.

1. *The method provides representations that allow the user to visualize the user interface* – The interface of the system is likely to be the most important component of the system to the end-user. As such, the ability to prototype the interface allows the end-user to accurately define the expectations at an early stage.

Guidance:

4 - Representations exist for visualizing user interfaces

2 - Representations exist for identifying interface components

0 - Representations not available

2. *The method models and assists in predicting the information processing and decision-making demands placed on the user* – Models showing the expected data processing and pointing out where the user will need to provide input is an important step, as it will allow the developer to design a system that will be well-balanced between dependencies on the user and on the system processing. With this type of information, system designers are able to load processors equally to ensure no one processor is a bottleneck to the entire system.

Guidance:

4 - Models for representing user interaction comprehensively defines requirements on user

2 - Models are available (such as behavior models) for representing user interaction

0 - Representations not available

3. *The method promotes the partitioning of user interface processing from other processing* – User interfaces are components of a system that may require changing for reasons none other than aesthetics. For this reason, a methodology promoting the partitioning of the interface from other processes will ensure a system that is not tied to its interface.

Guidance:

4 - Method enforces the separation of user interface components from other processing

2 - Method suggests separation of user interface components from other processing

0 - Method does not promote partitioning of user interface processing

The factors of this category are rated on the zero-to-four scale. For this evaluation consideration, the objective is to maximize the rating. A summary of the focus point is in the worksheet in Figure 22.

3.2.3 Building the Multiobjective Function

With the evaluation measures established, it is possible to build a multiobjective function. The multiobjective function is a normalized additive function that combines the proportional values of each of the fitness values for a particular consideration. The individual rating for each evaluation consideration is used to calculate a value using a single dimensional value function. This value is then multiplied by a weighting factor, which is determined by the decision-maker to represent the importance of the factor in the decision. The single dimensional objective factors and guidelines for determining weights are explored below.

EVALUATION CONSIDERATION: Interaction **OBJECTIVE:** Maximize Rating	
FACTOR	**RATING**
The method provides representations that allow the user to visualize the user interface	[0 1 2 3 4]
4 - Representations exist for visualizing user interfaces 2 - Representations exist for identifying interface components 0 - Representations not available	
The method models and assists in predicting the information processing and decision-making demands placed on the user	[0 1 2 3 4]
4 - Models for representing user interaction comprehensively defines requirements on user 2 - Models are available (such as behavior models) for representing user interaction 0 - Representations not available	
The method promotes the partitioning of user interface processing from other processing	[0 1 2 3 4]
4 - Method enforces the separation of user interface components from other processing 2 - Method suggests separation of user interface components from other processing 0 - Method does not promote partitioning of user interface processing	
TOTAL	/12

Figure 22: Interaction Worksheet

3.2.3.1 Single Dimensional Value Functions

The single dimensional value function provides a fitness value for a single evaluation consideration with regard to the alternative. In order for the multiobjective function to be normalized, the single dimensional value function must meet two properties. First, the lowest value the single dimensional function can return is zero. The second property is that the greatest value the function can return is one. This research makes use of two types of functions: exponential functions for the evaluation considerations that are of a continuous nature, and linear functions for those evaluation considerations that are discreet. Additionally, the exponential function is able to model different levels of significance a particular score may have. For example, in a monotonically decreasing exponential function, for one region of the function

the rating degrades very slowly, while in another region the rating degrades very quickly. The linear functions apply a uniform change in rating throughout the entire range.

3.2.3.1.1 Linear Functions

The evaluation considerations that use the zero-to-four rating scheme can be evaluated using a linear function. There are two versions of the linear function. The first is for the evaluation considerations with the objective of maximizing the rating. For these cases the function is:

$$v_i(X) = \frac{X}{\max_{\text{all factors in } i}(Rating)} \tag{1}$$

where *i* is the evaluation consideration. The second function is for evaluation considerations with the objective of minimizing the rating. In these cases, the function is:

$$v_i(X) = 1 - \frac{X}{\max_{\text{all factors in } i}(Rating)} \tag{2}$$

where *i* is the evaluation consideration. All of the proposed evaluation considerations requiring the linear function make use of the equation 1.

3.2.3.1.2 Exponential Functions

For evaluation considerations that can take on any range of values, the exponential function offers a simple way of normalizing the rating [18]. The exponential curve also allows for relative value increases for different ranges on the curve. The monotonically decreasing exponential function is:

$$v(x) = \begin{cases} \dfrac{1 - \exp(-(High - x)/\ldots)}{1 - \exp(-(High - Low)/\ldots)}, & \ldots \text{ Infinity} \\ \dfrac{High - x}{High - Low}, & \text{otherwise} \end{cases} \tag{3}$$

As the exponential constant $?$, approaches infinity, the function becomes linear. With the same constraint on $?$, the monotonically increasing exponential function is:

$$v(x) ? \begin{cases} ? \dfrac{1 ? \exp(? ?(x ? Low)/ ? ?)}{1 ? \exp(? ?(High ? Low)/ ? ?)} & ? ? Infinity \\ ? \dfrac{x ? Low}{High ? Low}, & otherwise \end{cases} \tag{4}$$

The evaluation consideration, *Cost of Acquiring Methodology and Support Tools*, is the only consideration being proposed that requires the exponential formula. As it has the objective of minimization of cost, it requires the monotonically decreasing version, equation 3. As this particular category is comparing the relative costs of the methodology for the organization, the variables of the equation can be determined based on this premise. Three variables, which the decision maker must provide, are required to solve the equation. Consider *High* to be the absolute maximum value the organization is willing to invest in the methodology. Next, consider *Low* to be the minimum value the organization will invest, probably zero dollars. In order to determine $?$, one more variable must be determined. Consider this variable to be the *target* investment that the organization is willing to make. It is the midpoint of the function where values to the left of the equation are more desirable than values to the right. This formulation limits the penalties against an alternative if it has a cost that is below the *target* investment. On the other hand, the penalty grows significantly faster for the alternatives that have costs greater than the *target* investment.

In order to calculate $?$, the three values—*target*, *high*, and *low*—must be specified. With these values, a normalized target, $z_{0.5}$, can be found by the equation:

$$\frac{(high ? target)}{(high ? low)} ? z_{0.5} \tag{5}$$

With a value determined for $z_{0.5}$, a normalized exponential constant, R, can be found. Unfortunately a closed form equation is not available for calculating R, so it can be determined by looking $z_{0.5}$ up in Figure 23. Thus, $?$ is calculated by:

$$? \quad ? \quad R? \; ?high ? low?$$

(6)

Now that ? is known, equation 3 can be completed.

$z_{0.5}$	R
0.00	--
0.01	0.014
0.02	0.029
0.03	0.043
0.04	0.058
0.05	0.072
0.06	0.087
0.07	0.101
0.08	0.115
0.09	0.130
0.10	0.144
0.11	0.159
0.12	0.174
0.13	0.189
0.14	0.204
0.15	0.220
0.16	0.236
0.17	0.252
0.18	0.269
0.19	0.287
0.20	0.305
0.21	0.324
0.22	0.344
0.23	0.365
0.24	0.387

$z_{0.5}$	R
0.25	0.410
0.26	0.435
0.27	0.462
0.28	0.491
0.29	0.522
0.30	0.555
0.31	0.592
0.32	0.632
0.33	0.677
0.34	0.726
0.35	0.782
0.36	0.845
0.37	0.917
0.38	1.001
0.39	1.099
0.40	1.216
0.41	1.359
0.42	1.536
0.43	1.762
0.44	2.063
0.45	2.483
0.46	3.112
0.47	4.157
0.48	6.243
0.49	12.497

$z_{0.5}$	R
0.50	Infinity
0.51	-12.497
0.52	-6.243
0.53	-4.157
0.54	-3.112
0.55	-2.483
0.56	-2.063
0.57	-1.762
0.58	-1.536
0.59	-1.359
0.60	-1.216
0.61	-1.099
0.62	-1.001
0.63	-0.917
0.64	-0.845
0.65	-0.782
0.66	-0.726
0.67	-0.677
0.68	-0.632
0.69	-0.592
0.70	-0.555
0.71	-0.522
0.72	-0.491
0.73	-0.462
0.74	-0.435

$z_{0.5}$	R
0.75	-0.410
0.76	-0.387
0.77	-0.365
0.78	-0.344
0.79	-.0324
0.80	-0.305
0.81	-0.287
0.82	-0.269
0.83	-0.252
0.84	-0.236
0.85	-0.220
0.86	-0.204
0.87	-0.189
0.88	-0.174
0.89	-0.159
0.90	-0.144
0.91	-0.130
0.92	-0.115
0.93	-0.101
0.94	-0.087
0.95	-0.072
0.96	-0.058
0.97	-0.043
0.98	-0.029
0.99	-0.014

Figure 23: Normalized Exponential Constants [18]

3.2.3.2 Weights

There are two issues that go into determining the weights for the evaluation considerations. The first is developing a relative rating of importance based on the consideration. This section further explores the evaluation considerations and provides some guidelines. The second issue, taking the relative weights and normalizing the values for appropriate use with the tool, is covered in Section 4.2.1.

Determining the impact each evaluation consideration measure has on the problem has the greatest effect on the choice the decision-maker must make. In an ideal world, the assessor would only be required to rate a methodology once. That is, it is not likely that methodology is going to change, and if it does, it

should be evaluated as a new methodology. As such, the impact that the software project will have on the decision lies with the importance of each consideration category.

3.2.3.3 Multiobjective Function

With the details in place for determining the weights and scores for each evaluation consideration, a multiobjective function can be formed. The multiobjective function for the methodology selection problem is given by:

$$V(X) = \sum_{i} w_i v_i(X_i) \quad \text{where } i = \text{cost, bus, mat, reuse, ent, dis, env, ar, dss, int} \tag{7}$$

The mapping of the index values to the evaluation consideration can be found in Figure 24.

Index	Evaluation Consideration
cost	Cost of Acquiring Methodology and Support Tools
bus	Organizational Business Practices
mat	Methodology Maturity
reuse	Availability of Reusable Components
ent	Enterprise Integration
dis	Distribution
env	Environment
ar	Agility and Robustness
dss	Dynamic Structure and Scalability
int	Interaction

Figure 24: Index Mapping

3.2.4 Developing a Software Implementation

Developing the software implementation requires a process for the implementation to model, as well as an implementation strategy. Section 3.2.4.1 outlines the process that was modeled by the implementation described in Section 3.2.4.2.

3.2.4.1 Applying the Decision Tool

With the framework of the decision analysis tool and the details determined, the tool is ready to be used to provide a decision-maker with a fitness value for any number of methodology alternatives for a particular software problem. The four distinct steps in applying the decision analysis tool are:

1. Determine the weights of the evaluation considerations
2. Score each methodology based on the answers to the focus points
3. Calculate the multiobjective function for each methodology
4. Select the methodology for software development

Each of these steps is further explained in Chapter IV. Additionally, Chapter IV provides examples of the analysis tool application on a number of software requirements.

3.2.4.2 Automation of Decision Analysis Tool

In order to make the Decision Analysis Tool more useful to the decision maker, a software implementation of the process was required. In addition to calculating a number of single dimension value functions and the multiobjective function for each alternative, the ability to make a decision on an appropriate software engineering methodology requires a number of calculations to test the sensitivity of the assumptions the decision maker makes when specifying the weights for each of the evaluation considerations.

The Decision Analysis Tool was implemented as a Java Applet. The applet allows for easy distribution to a number of user platforms. It can be used by itself, or inserted into a web page. Using an applet also allows for the implementation of a graphical user interface for data gathering and control of the execution flow of the computations. Additionally, the tool can be executed as an application, allowing the user to write data to disk for analysis by an auxiliary application.

The interface is designed around the steps described in Section 3.2.4.1, with each distinct step being an individual panel in the applet. Though the analysis process discussed within Section 3.2.4.1 is generalized for comparing any number of methodologies, the application is designed to analyze two alternatives at a time. In order to rate the alternatives, the focus points in the worksheets in Figures 9-17 have been incorporated into the interface. In addition to the Java applet, a data analyzer was developed in a Microsoft Excel Workbook. The data analyzer was designed to leverage the software packages graphing functionality. A user's manual for the installation and operation of the Software Engineering Methodology Decision Analysis Tool and Data Analyzer is contained in Appendix D.

3.3 Summary

This chapter applied the decision-making framework, which was discussed in Chapter II, to the problem of selecting an appropriate software engineering methodology for a particular problem. Beginning with a problem statement, a generalized value hierarchy was developed based on three areas of methodology characterizations.

After determining the evaluation considerations that make up the value hierarchy, the details of the evaluation measures are defined. This research is using a set of criteria-based questions, or focus points, which the decision-maker answers about each method, to determine a rating for the category. These ratings are used in single-dimensional objective functions that, when combined with the weights of the particular category, are used to determine a multiobjective fitness value.

This chapter ends with a description of the process to apply the decision tool to software methodology decision problems and a software implementation that was developed to automate the calculations. Next, Chapter IV expands the application process described in Section 3.2.4.1 and demonstrates the application of the tool on different software problems.

IV. Decision Analysis Application

4.1 Introduction

The first step in defining the decision making process for the Software Methodology Selection problem was to select an appropriate framework as described in Chapter II. Next, in Chapter III, the framework selected, in this case Multiobjective Decision Analysis, was developed into a Decision Analysis Tool to include a value hierarchy based on the opinions of the members of the software engineering and multiagent system communities, focus point rating systems for evaluation of the considerations in the value hierarchy, and objective functions for computing ratings for comparison.

This chapter demonstrates the application of the Decision Analysis Tool. First, Section 4.2 refines the application process defined in Section 3.2.4.1. Next, Section 4.3 describes the alternative methodologies used for the demonstration. Several software requirements are analyzed step-by-step in Section 4.4. Then, Section 4.5 describes the analysis conducted on the assumptions made during the decision analysis.

4.2 Decision Analysis Application Process

This section refines the four-step process to application described in Section 3.2.4.1. The four steps are: determine weights, rate evaluation measures, compute multiobjective function, and select a methodology for the software requirement problem.

4.2.1 Determining Weights

This phase of the application of the decision analysis tool is focused on transforming the weights, which are relative to each other into weights that are normalized to total one. The first step is for the user to determine which evaluation considerations are not important to the particular decision. The weights of these considerations can be set to zero. The user, then, ranks the level importance of each evaluation consideration. This step determines the level of importance each factor has on the decision by ordering the

considerations in successive level of importance. Next, the technique described in Section 2.3.1.3 is applied to normalize the weights.

4.2.2 Rating Evaluation Measures

The next phase in applying the decision analysis tool is to rate the alternatives based on the criteria-based questions developed for each evaluation consideration. Though many of the questions require a qualitative rating, it is important to remember that multiple alternatives are being compared. To the best of the decision-makers ability, the scores for each question should be relative to the other alternatives.

4.2.3 Computing Multiobjective Function

The final phase in the application of the decision analysis tool is the computation of the multiobjective function. The data gathered in the previous phases is the heart of this step. The derived values for the weights and the ratings are used to generate a single normalized fitness value through variable substitution.

4.2.4 Methodology Selection

After completing the steps of the decision analysis tool, the decision maker has a quantitative rating on each of the alternative methodologies. These ratings are based on the methodology's fitness with respect to each of the appropriate evaluation considerations as well as the decision maker's biases on those considerations through the weights. However, there is still an issue of tradeoffs involved with the methodologies that may not be captured in the given framework.

As an example, take the issue of distribution. In a problem that does not involve distribution of any sort, the decision maker would weight the category with a zero. Completing the decision analysis tool may yield results that a particular agent-oriented approach rates higher than a particular object-oriented approach. The tool was not able to capture the tradeoff that the agent-oriented approach develops the problem the same way, whether it is distributed or not. The implementation of the agent-oriented approach

introduces a number of performance problems because of the overhead caused by the distributed communication features inherent to the methodology.

As tradeoffs such as this are evident in the decision-making problem, the decision maker must be cognizant of the fact and attempt to weigh the impact into the decision. After all of the considerations are made, the decision maker may then make a well-informed decision about which software engineering methodology is appropriate for the software requirement.

4.3 Software Engineering Methodology Alternatives

In order to validate the Decision Analysis Tool, specific alternative software engineering methodologies must be evaluated for a set of problems. Although a large number of methodologies exist in paradigms such as Object-Oriented, Agent-Oriented, Formal Methods, Functional Decomposition, Component-ware, etc., three alternatives were evaluated for a set of four software requirement problems. For the set of problems, MaSE, Booch's Object Oriented Analysis, and Design and Yourdon's Modern Structured Analysis were selected as representative methodologies for agent-oriented, object-oriented, and traditional software engineering, respectively [2, 39, 43]. These methodologies define models and representations to deal with the analysis and design phases of software development. Additionally, CASE tools exist to support each method.

4.4 Applications of the Decision Analysis Tool

Four example systems have been selected to demonstrate the application of the Decision Analysis tool. These systems represent the following application domains: information systems, real-time systems, and intelligent systems. The applications have been selected for their particular representation of a particular methodology. As each was developed when different methodologies were popular, the tool validates the decision to use the particular approach.

It is assumed that the software engineer is a member of a fledgling software development organization in order to test the management issues of the problem. Also, it is assumed that the engineer has been authorized $6,000 for the acquisition of methodology and tools.

4.4.1 Rating Considerations

The second step in the process of applying the decision analysis tool is determining the scores for each of the relevant evaluation considerations. This step will be completed out of order because each of the case studies below will use this data. The three alternative methodologies selected in Section 4.2 will be evaluated for all of the evaluation considerations. Under a normal application, only the relevant considerations would be evaluated. The evaluation consideration ratings are determined by addressing the focus points presented in Chapter III. For this step, the appropriate worksheets from Figures 11 through 20 are used. The order of the rating is not important, so the considerations are rated in the order they are found in Chapter III. *Cost of Acquiring Methodology and Support Tools* is the first evaluation consideration rated.

4.4.1.1 Cost of Acquiring Methodology and Support Tools

Rating the *Cost* evaluation consideration first required some data gathering. As the responses for this evaluation consideration require real costs for training classes, reference material, and such, several resources were considered. Additionally, the tool support that was selected for MaSE and Booch—agentTool and Rational Rose, respectively—were selected based on the availability to the author and the price currently being charged. For the Yourdon methodology, specific CASE tools were not available, so a tool providing the same level of support as agentTool and Rose was selected. Prices reflect a single-user license agreement. The summary of the costs is shown in Figure 25.

EVALUATION CONSIDERATION: Cost of Acquiring Methodology and Support Tools **OBJECTIVE:** Minimize cost			
FACTOR	**MaSE RATING**	**Booch RATING**	**Yourdon RATING**
The cost to train personnel to use the methodology	$555	$535	$535
The cost of additional reference material on the methodology	0	$30	$55
The cost of the software development tool(s)	0	$4290	$3995
The cost of maintenance on the product for the expected duration of use	0	0	0
The cost to train personnel to use the tool(s)	$1110	$574	$2000
The cost to install the tool(s)	$25	$25	$25
The cost of additional hardware required by the specific tool(s)	0	0	0
The cost of additional software required by the specific tool(s)	0	0	0
TOTAL	$1690	$5454	$6610
Maximum Allowable Expenditure	$8000		
Preferred Expenditure	$6000		
Minimum Allowable Expenditure	0		

Figure 25: Cost of Acquiring Methodology and Support Tools Worksheet Summary

In addition to the data collected for the *Cost*, it is necessary to determine the value of *?*. Based on the formula given in Equations 5 and 6, $z_{0.5}$ is first needed to find *R*:

$$z_{0.5} = (high - target) / (high - low)$$

$$= (8000 - 6000) / (8000 - 0)$$

$$= .25$$

Using the table in Figure 23, the value of *R* corresponding to $z_{05} = 0.25$ is 0.410. With the value of *R* known, Equation 6 calculates *?*:

$$? = R \times (high - low)$$

$$= 0.410 \times (8000 - 0)$$

$$= 3280$$

With *?*, *high*, and *low*, the exponential single dimensional value function in Equation 3 is used for determining the fitness of the evaluation consideration.

From the data that is shown in Figure 25, the experimental MaSE is significantly less expensive than the more mature and commercially refined methodologies. A counterpoint to this detail is the evaluation consideration *Methodology Maturity*. This evaluation consideration, in Section 4.3.1.3, scores the methodology that has been accepted by industry higher than the experimental alternatives.

4.4.1.2 Organizational Business Practices

The next evaluation consideration measured is *Organizational Business Practices*. Based on the assumption made above, the evaluation consideration is rated as if a project management process existed with the business practices of the fictional organization. Generally speaking, the three methodologies rate similarly. They provide a well-defined methodology for the development of a software project, but the Booch methodology specifically addresses software development from a micro- and macro-development level, hence the higher rating [2]. As for the availability of techniques to analyze the progress of the development, none of the methodologies provide support.

Partitioning the development system is an area in which MaSE did not score as high as the other methodologies because its components are very dependent upon communication protocols with other components. This affects the ability to divide up the work in base units. MaSE does rate higher in the ability to rapidly generate high-level representations for feasibility analysis. In fact, the tool support for MaSE provides verification testing to ensure the design is free from communication deadlock. The other focus points of this consideration rated the same for each consideration. The summary of the methodologies' ratings for this category is shown in Figure 26.

EVALUATION CONSIDERATION: Organizational Business Practices **OBJECTIVE:** Maximize Rating			
FACTOR	**MaSE RATING**	**Booch RATING**	**Yourdon RATING**
The method provides planning techniques that lead to milestone definitions and project plans that are consistent with use of the method and the implementation language	2	3	2
The method has analysis techniques that can be used during reviews or that can help you gauge progress during development	0	0	0
Representations are clear and easy enough to understand to be used for design reviews	3	3	3
Ability to rapidly develop high-level design representations that can be analyzed to determine the most feasible design approach	3	2	2
The method helps partition the system into manageable pieces that can be given to development team members	3	4	4
The method establishes and enforces well-defined representation standards for each view at each stage of development	4	4	4
The method provides hierarchical forms for all types of representations	4	4	4
TOTAL	19/28	20/28	19/28

Figure 26: Organizational Business Practices Worksheet Summary

4.4.1.3 Methodology Maturity

As noted above, the *Methodology Maturity* consideration is the counterpoint to the *Cost* consideration. There are only a few examples of software that is used commercially but available at little expense—Java comes to mind—so, for the most part, if a methodology and tools can be acquired for little to nothing, it is likely that it is immature. Industrial strength CASE tools, generally come with an industrial strength price tag. Also, as a methodology becomes more mature, training and reference material for the methodology are more expensive, as the training is being done by technology training academies and the reference is in the form of books rather than just academic papers. In this case, MaSE rated lower than the other alternatives because it is a methodology that is still being developed. The summary of the scores is in Figure 27.

EVALUATION CONSIDERATION: Methodology Maturity **OBJECTIVE:** Maximize Rating			
FACTOR	**MaSE RATING**	**Booch RATING**	**Yourdon RATING**
Support tools are available for the methodology	2	4	4
Training/reference material is available for the methodology	1	4	4
Systems developed with the methodology exist	1	4	4
TOTAL	4/12	12/12	12/12

Figure 27:Methodology Maturity Summary

4.4.1.4 Reusable Components

The final management evaluation consideration is *Reusable Components*. In this case, MaSE and Booch allow for the incorporation of reusable components, though the support is greater in Booch. In fact, the models used in later OMG versions of UML are derived from Booch's module diagrams [25]. The support MaSE provides is ambiguous as there is no differentiation between a developed component and a newly designed component. Component reuse in structured analysis is more complicated. A system designed in this paradigm is highly cohesive and coupled, making it difficult to separate any one subsystem from the entire system. Figure 28 summarizes the ratings for *Reusable Components*.

EVALUATION CONSIDERATION: Reusable Components **OBJECTIVE:** Maximize Rating			
FACTOR	**MaSE RATING**	**Booch RATING**	**Yourdon RATING**
The method encourages separating requirements into essential and negotiable classes, where negotiable classes identify the opportunity for reuse	0	2	0
The method encourages designers to suggest alternatives provided by existing components to negotiable requirements	2	3	0
TOTAL	2/8	5/8	0/8

Figure 28: Reusable Components Worksheet Summary

4.4.1.5 Legacy System Integration

Legacy System Integration is the first technical consideration evaluated. Legacy Systems are treated, for the most part, as reusable components, though components are more specific in functionality. For an agent system, MaSE would represent the system with an agent wrapper that would be developed specifically to interact with the legacy system. The rest of the new system would interact with the agent

wrapper. The Booch methodology, on the other hand, supports the legacy system integration, again, through the module diagram. Due to the strong coupling and cohesion, the Yourdon methodology makes system integration difficult. Systems outside of the new system are represented as terminators in dataflow diagrams [43]. The summary of the ratings for *Legacy System Integration* is in the worksheet in Figure 29.

EVALUATION CONSIDERATION: Legacy System Integration **OBJECTIVE:** Maximize Rating			
FACTOR	**MaSE RATING**	**Booch RATING**	**Yourdon RATING**
Methodology provides representations to allow the incorporation of a legacy system if it has a documented interface	3	2	0
Methodology provides a representation for referencing legacy system functions/methods	2	2	1
TOTAL	5/8	4/8	1/8

Figure 29: Legacy System Integration Worksheet Summary

4.4.1.6 Distribution

Distribution is the next evaluation consideration scored. In this case, the MaSE methodology clearly led the pack. This is in part because of the nature of agent systems. When developed, agents interact with each other through a message passing system. As such, the agents are not coupled until a communication needs to take place. There is no requirement for the agents to reside on the same systems. Objects, on the other hand, have more stringent rules. The underpinnings of a message-based communication system are not inherent to the object system. Both methodologies support the use of a deployment diagram for specifying system configurations, but MaSE's diagram is used to specify the actual location of the agents, where as the Booch version is used to map processing and resources. Yourdon's support of distribution is vague. Again, terminators in the dataflow diagram can be distributed, but there is not a specific representation that captures this type of information. Figure 30 summarizes the scores for the three methodologies with regard to support of distribution.

EVALUATION CONSIDERATION: Distribution **OBJECTIVE:** Maximize Rating			
FACTOR	**MaSE RATING**	**Booch RATING**	**Yourdon RATING**
The method provides a representation for specifying system network configuration	3	2	1
The method provides a representation for specifying deployment of entities throughout the network	4	2	0
The method provides the ability to specify locations of resources throughout the system network	2	2	0
TOTAL	9/12	6/12	1/12

Figure 30: Distribution Worksheet Summary

4.4.1.7 Environment

The next factor in the decision is the *Environment*. In this case, all of the methodologies can be transformed into an implementation programming language supported on the various platforms required by the case studies. Object-oriented languages such as Java support the MaSE and Booch methods more readily than the functional techniques like Yourdon. As Java is portable to different operating systems, the methodologies score well. The deployment diagrams assist code developers in recognizing the potential locations of the code, so it is possible to develop the system to handle any operating system it resides on. The scores for *Environment* are captured in Figure 31.

EVALUATION CONSIDERATION: Environment **OBJECTIVE:** Maximize Rating			
FACTOR	**MaSE RATING**	**Booch RATING**	**Yourdon RATING**
The programming language for implementation is supported by the software/hardware systems on which the program will be deployed	4	4	4
The methodology supports deployment modeling in which the engineer can easily determine necessary changes in the case of system specific commands	2	2	0
The programming languages supported by the methodology for implementation requires little additional time for compilation for each system type	4	4	0
TOTAL	10/12	10/12	4/12

Figure 31: Environment Worksheet Summary

4.4.1.8 Agility and Robustness

Agility and Robustness is the next consideration rated. This is a category in which the agent-oriented methodology rated higher than the other two methodologies. This is because the nature of the agent system is to handle communication with agents that may not be known at design time. The behavior models are able to represent expected exceptional behavior. The summary of the ratings for *Agility and Robustness* is in Figure 32.

EVALUATION CONSIDERATION: Agility and Robustness **OBJECTIVE:** Maximize Rating			
FACTOR	**MaSE RATING**	**Booch RATING**	**Yourdon RATING**
The methodology's representations provide the ability to model normal processing and exception processing	3	3	2
The method provides techniques to model dynamic system reconfiguration	2	0	0
The method provides techniques to analyze system performance for all configurations	0	0	0
TOTAL	5/12	3/12	2/12

Figure 32: Agility and Robustness Worksheet Summary

4.4.1.9 Dynamic Structure and Scalability

The next evaluation consideration scored is *Dynamic Structure and Scalability*. In this case the agent- and object-oriented methodologies scored higher than the Yourdon. The reason the Yourdon methodology scored so low, again, is related to the coupling and cohesion of functions. The other methodologies, however, allow systems to be developed in ways such that resources can be added and deleted without shutting the system down. MaSE, for example, uses the agent wrapper concept to interface with resources throughout the system. In this case, the agent that corresponds with the resource wrapper agents must learn about the existence of the wrapper agents. Some knowledge source must be available for the initial communications to work. This may happen through an agent that acts as a registration authority, brokers, or, in the case of closed systems, static reference to particular agents. Systems developed following the Booch methodology can also be designed to accept lists of resources, perhaps at start-up or during execution, in order to make dynamic connections. The summary of the *Dynamic Structure and Scalability* consideration is in Figure 33.

EVALUATION CONSIDERATION: Dynamic Structure and Scalability **OBJECTIVE:** Maximize Rating			
FACTOR	**MaSE RATING**	**Booch RATING**	**Yourdon RATING**
The methodology provides techniques for dynamic integration of new subsystems/resources with minimal disruption to existing system	3	2	0
The methodology provides techniques for the removal of existing subsystems/resources with minimal disruption to remaining system	3	3	0
TOTAL	6/8	5/8	0/8

Figure 33: Dynamic Structure and Scalability Worksheet Summary

4.4.1.10 Interaction

For the *Interaction* evaluation consideration, the three alternatives rated similarly, though Booch rated higher. The Booch methodology is the only one that specifically addressed the question of developing a concept for the user interface as well as partitioning the user interface processing from the other processing. All of the methodologies provide support in predicting the decision-making demands on the user through the respective behavior models. The major difference between the three methodologies is the extent to which the Booch method suggests partitioning. The Booch methodology promotes placing all interface components in separate classes [2]. The Yourdon methodology suggests placing the interface requirements in procedures, but these are generally coupled with the respective modules with which they are used [43]. MaSE suggests modeling user roles separate from other roles in the system, but there is not much guidance with regard to developing the interface as part of an agent's set of internal components. The final summary of ratings is shown in Figure 34.

EVALUATION CONSIDERATION: Interaction **OBJECTIVE:** Maximize Rating			
FACTOR	**MaSE RATING**	**Booch RATING**	**Yourdon RATING**
The method provides representations that allow the user to visualize the user interface	0	2	0
The method models and assists in predicting the information processing and decision-making demands placed on the user	4	4	4
The method promotes the partitioning of user interface processing from other processing	2	4	2
TOTAL	6/12	10/12	6/12

Figure 34: Interaction Worksheet Summary

4.4.2 Software Requirement Case Studies

This section describes the four case studies evaluated for the application of the decision analysis tool. In addition to information provided below for each case, the assumption in Section 4.4 also holds.

4.4.2.1 Case 1: Content Search

The first software requirement example is for an information management system called Content Search System; the requirement statement is shown in Figure 35. The concept of this system was originally developed to compare the performance capabilities of dynamic and static multiagent systems [29]. In this case, the system requirements are being used to compare the agent-oriented and object-oriented methodologies.

The requirement of the system is to provide enterprise search capability to an organization of networked users. The organization has determined that the most feasible data storage technique is to use the large capacity hard drives on the personal computers and workstations within the organization, rather than invest in a single shared data storage system. The only requirement of the system is to determine the location of files that may have information a user requires. Systems are designated to belong to certain functional areas, so users should be able to restrict the search to a single functional area. With these requirements in mind, the first step in selecting an appropriate methodology for development is determining which factors are relevant to the decision and to weight them accordingly.

4.4.2.1.1 Weighting the Evaluation Considerations

The first step in weighting the evaluation considerations is to determine which of the evaluation considerations are relevant to the particular problem. Below are each of the categories and the analysis decision made as to whether or not the consideration is relevant.

- *Cost of Acquiring Methodology and Support Tools* – Relevant. The approach to this problem is that the software engineer is part of an organization that is looking to adopt the methodology and supporting tools.

- *Organizational Business Practices* – Relevant. Although the software engineer is the only employee in the fledgling department, the engineer does have the responsibility of providing project updates to other interested parties outside of the department.

Content Search System Requirements

Organization X is responsible for collecting vital data on a variety of subjects. A business practice that the organization follows is teamwork, where information is shared freely. The amount of information collected would necessitate a large data storage area. In order to reduce costs, the organization would like to make use of the fixed storage devices in each employee's local personal computer. This policy, however, makes it difficult to share the information.

The problem is to design a software system that will allow users to locate sources of information throughout the organization. Details include:

Searching – The user must be able to search the documents for specified key words or phrases

Reporting – The system should report the results of the search in an order that would suggest higher priority to more relevant documents

Documents – The documents that are generated by the employees are simple text documents (no formatting)

Results – The information included in the results should include the location of the document, the name of the document, and the number of occurrences of the key words or phrases

Scope – As there are many functional areas within the organization, the user should be able to specify the particular functional areas that need to be examined

Environment – The computer environment that the employees' use is a mix of PCs running Windows operating systems and Unix workstations

Figure 35: Content Search System Requirements

- *Methodology Maturity* – Relevant. The decision to change to a new methodology will require some degree of evidence that it will produce quality software.

- *Integration of Reusable Components* – Irrelevant. A library of reusable components is not available to the software engineer.

- *Legacy System Integration* – Irrelevant. The system is not required to incorporate any existing software systems.

- *Distribution* – Relevant. The users of the system will require access from different nodes on the network. Likewise, the data that the users will require is stored on many hard drives throughout the network.

- *Environment* – Relevant. The environment of the network is a mixture of Sun Workstations running Solaris OS and Personal Computers running Windows NT.

- *Agility and Robustness* – Relevant. The users of the system will expect predictability and reliability.

- *Dynamic Structure and Scalability* – Relevant. The organization is growing and as new employees are hired, the hardware systems they are given will need to be linked to the software system for access and data storage.

- *Interaction* – Relevant. The system must provide an interface for the user to submit requests.

Following this coarse partitioning of the evaluation considerations, the next step is to refine the ranking by determining each evaluation considerations level of importance. The most important level of evaluation considerations consists of those that rate issues that are constraints of the software requirement problem itself: *distribution*, *environment*, *interaction* and *cost of methodology and support tools*. The next level of ranking is for the evaluation considerations that are important to the success of the project, but the level of achievement may be negotiable. A*gility and robustness* and *dynamic structure and scalability* have been assigned to this level. For this problem, *organizational business practices* and *methodology maturity* considerations were placed in a third level, as a goal of the departments is to develop a well-defined and repeatable software engineering process.

The next step in refinement is determining the relative weighting of each evaluation consideration with regard to the next least important consideration. For the evaluation considerations in the same rank tier, the relative weight is one. It is only necessary to determine the boundary relative weight, that is the last consideration in the higher rank relative to the first consideration in the next lower rank. The

considerations selected are done so, more or less, from any ordering of the considerations. In this case, the evaluation considerations have been ordered alphabetically, so *interaction* is compared to *agility and robustness*, and *dynamic structure and scalability* is compared to *organizational business practices*. The tier of evaluation considerations that measure the constraints of the problem was determined by the decision maker to be twice as important as the lower rank. Likewise, the tier of negotiable considerations was determined to be one and a quarter times important as the lowest tier.

With the relative weighting determined, the next step is to determine the normalized weights for each evaluation consideration following the method described in Section 2.3.1.3. For illustration, the algebraic calculations are shown below using the subscripts as defined in Figure 24.

$$w_{mat} = w_{org}$$

$$w_{dss} = 1.25 \cdot (w_{org})$$

$$w_{ar} = 1 \cdot (w_{dss}) = 1 \cdot (1.25 \cdot w_{org})$$

$$w_{int} = 2 \cdot (w_{ar}) = 2 \cdot (1 \cdot (1.25 \cdot w_{org}))$$

$$w_{env} = 1 \cdot (w_{int}) = 1 \cdot (2 \cdot (1 \cdot (1.25 \cdot w_{org})))$$

$$w_{dis} = 1 \cdot (w_{env}) = 1 \cdot (1 \cdot (2 \cdot (1 \cdot (1.25 \cdot w_{org}))))$$

$$w_{cost} = 1 \cdot (w_{dis}) = 1 \cdot (1 \cdot (1 \cdot (2 \cdot (1 \cdot (1.25 \cdot w_{org})))))$$

Setting the sum of the weights to 1,

$$1 = w_{org} + w_{mat} + w_{dss} + w_{ar} + w_{int} + w_{env} + w_{dis} + w_{cost}$$

$$= w_{org} + w_{org} + (1.25 \cdot w_{org}) + (1.25 \cdot w_{org}) + (2 \cdot (1 \cdot (1.25 \cdot w_{org})))$$
$$+ (2 \cdot (1 \cdot (1.25 \cdot w_{org}))) + (2 \cdot (1 \cdot (1.25 \cdot w_{org}))) + (2 \cdot (1 \cdot (1.25 \cdot w_{org})))$$

$$= w_{org} \cdot (1 + 1 + 1.25 + 1.25 + 2.5 + 2.5 + 2.5 + 2.5)$$

$$= w_{org} \cdot (14.5)$$

$$w_{org} = 1 / 14.5 = 0.069$$

Hence,

$w_{mat} = 0.069$

$w_{dss} = 1.25 ? (0.069) = 0.086$

$w_{ar} = 1 ? (0.086) = 0.086$

$w_{int} = 2 ? (0.086) = 0.172$

$w_{env} = 1 ? (0.172) = 0.172$

$w_{dis} = 1 ? (0.172) = 0.172$

$w_{cost} = 1 ? (0.172) = 0.172$

With the weights determined for the relevant evaluation considerations, the first step of the decision-making process is complete. The results of this step are summarized in Figure 36.

Rank	Evaluation Consideration	Relative Weight	Normalized Weight
1	Cost of Methodology & Support Tools	1	0.172
1	Distribution	1	0.172
1	Environment	1	0.172
1	Interaction	2	0.172
2	Agility and Robustness	1	0.086
2	Dynamic Structure and Scalability	1.25	0.086
3	Methodology Maturity	1	0.069
3	Organizational Business Practices		0.069

Figure 36: Content Search Weighting Summary

4.4.2.1.2 Rating the Evaluation Considerations

The next step of the decision analysis is to score the evaluation considerations. The focus points are examined and scored for each of the evaluation considerations that have been determined to be relevant in the weighting step. This step is documented in Section 4.3.1. The single dimensional value functions for case 1 are summarized in Figure 37.

Evaluation Consideration	SDVF Fitness – MaSE	SDVF Fitness – Booch	SDVF Fitness – Yourdon
Cost of Methodology & Support Tools	0.937	0.591	0.378
Distribution	0.750	0.500	0.083
Environment	0.833	0.833	0.333
Interaction	0.500	0.833	0.500
Agility and Robustness	0.417	0.250	0.167
Dynamic Structure and Scalability	0.750	0.625	0
Methodology Maturity	0.333	1.000	1.000
Organizational Business Practices	0.679	0.714	0.679

Figure 37: Single Dimensional Value Function Fitness Values-Case1

4.4.2.1.3 Calculating the Multiobjective Value Functions

To complete the decision analysis of the content search example, the multiobjective value function is calculated. To accomplish this step, the single dimensional value functions are multiplied by the appropriate weight and summed. For the MaSE alternative, the multiobjective value is determined by:

$$\begin{aligned}\mathbf{V_{MaSE}(X)} = {} & w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{mat}v_{mat}(x_{mat}) + w_{dis}v_{dis}(x_{dis}) \\ & + w_{env}v_{env}(x_{env}) + w_{ar}v_{ar}(x_{ar}) + w_{dss}v_{dss}(x_{dss}) + w_{int}v_{int}(x_{int}) \\ = {} & 0.172\, v_{cost}(1690) + 0.069\, v_{org}(19) + 0.069\, v_{mat}(4) + 0.172\, v_{dis}(9) \\ & + 0.172\, v_{env}(10) + 0.086\, v_{ar}(5) + 0.086\, v_{dss}(6) + 0.172\, v_{int}(6) \\ = {} & 0.161 + 0.047 + 0.023 + 0.129 + 0.143 + 0.036 + 0.065 + 0.086 \\ = {} & \mathbf{0.689}\end{aligned}$$

The Booch alternative is calculated to be:

$$\begin{aligned}\mathbf{V_{Booch}(X)} = {} & w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{mat}v_{mat}(x_{mat}) + w_{dis}v_{dis}(x_{dis}) \\ & + w_{env}v_{env}(x_{env}) + w_{ar}v_{ar}(x_{ar}) + w_{dss}v_{dss}(x_{dss}) + w_{int}v_{int}(x_{int}) \\ = {} & 0.172\, v_{cost}(5454) + 0.069\, v_{org}(20) + 0.069\, v_{mat}(12) + 0.172\, v_{dis}(6) \\ & + 0.172\, v_{env}(10) + 0.086\, v_{ar}(3) + 0.086\, v_{dss}(5) + 0.172\, v_{int}(10) \\ = {} & 0.102 + 0.049 + 0.069 + 0.086 + 0.143 + 0.022 + 0.054 + 0.143 \\ = {} & \mathbf{0.668}\end{aligned}$$

The Yourdon alternative is calculated to be:

$$V_{Yourdon}(X) = w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{mat}v_{mat}(x_{mat}) + w_{dis}v_{dis}(x_{dis}) + w_{env}v_{env}(x_{env}) + w_{ar}v_{ar}(x_{ar}) + w_{dss}v_{dss}(x_{dss}) + w_{int}v_{int}(x_{int})$$

$$= 0.172\, v_{cost}(6610) + 0.069\, v_{org}(19) + 0.069\, v_{mat}(12) + 0.172\, v_{dis}(1) + 0.172\, v_{env}(4) + 0.086\, v_{ar}(2) + 0.086\, v_{dss}(0) + 0.172\, v_{int}(6)$$

$$= 0.065 + 0.047 + 0.069 + 0.014 + 0.057 + 0.014 + 0 + 0.086$$

$$= \mathbf{0.353}$$

4.4.2.1.4 Determining the Best Alternative

Based on the above calculations and assumptions, the decision analysis tool has determined the MaSE and Booch methodologies to be the better of than the Yourdon methodology. With regard to the MaSE and Booch methodologies, MaSE is rated 0.021 points higher than the Booch methodology. Considering the small margin of difference between these two methodologies, the decision-maker is advised to contemplate results of a sensitivity analysis. The sensitivity analysis tests the results by systematically adjusting the weight of one evaluation consideration while maintaining the ratio of the weights of the other evaluation considerations. The sensitivity analysis is documented in Section 4.5.

4.4.2.2 Case 2: Weather Monitoring Station

The second example system evaluated is a weather monitoring system. This system is a case study for object-oriented development following the Booch methodology [2]. It is suggested that either object-oriented or structured analysis are appropriate techniques for designing this system, but the problem "lends itself well to an object-oriented architecture" [2]. In this case, the object-oriented approach is being compared to a potential agent-oriented approach.

One interesting factor that sets this problem apart from the previous is the requirement for specific data acquisition. Additional details on the system requirements include specific sampling rates of each of the sensors. Managing this real-time data collection requirement is an important aspect of the problem. A requirements statement for this problem is in Figure 38.

4.4.2.2.1 Weighting the Evaluation Considerations

To begin selecting a methodology, the first step is weighing the evaluation considerations. To do this, the relevant evaluation considerations need to be determined. Below are each of the categories and the analysis decision made as to whether or not the consideration is relevant.

Weather Monitoring Station Requirements

This system shall provide automatic monitoring of various weather conditions. Specifically, it must measure:

- Wind speed and direction

- Temperature

- Barometric pressure

- Humidity

The system shall also provide the following derived measurements:

- Wind chill

- Dew point temperature

- Temperature trend

- Barometric pressure trend

The system shall have a means of determining the current time and date, so that it can report the highest and lowest values of any of the four primary measurements during the previous 24 hour period.

The system shall have a display that continuously indicates all eight primary and derived measurements, as well as the current time and date. Through the use of a keypad, the user may direct the system to display the 24-hour high or low value of any one primary measurement, together with the time of the reported value.

The system shall allow the user to calibrate its sensors against known values, and to set the current time and date.

Figure 38: Weather Monitoring Station Requirements [2]

- *Cost of Acquiring Methodology and Support Tools* – Relevant. The approach to this problem is that the software engineer is a part of an organization that is looking to adopt the methodology and supporting tools.

- *Organizational Business Practices* – Relevant. Although the software engineer is the only employee in the fledgling department, the engineer does have the responsibility of providing project updates to other interested parties outside of the department.
- *Methodology Maturity* – Relevant. The decision to change to a new methodology will require some degree of evidence that it will produce quality software.
- *Integration of Reusable Components* – Irrelevant. A library of reusable components is not available to the software engineer.
- *Legacy System Integration* – Irrelevant. The system is not required to incorporate any existing software systems.
- *Distribution* – Irrelevant. Based on the requirements of this system, the sensors will use memory shared with the system for the storing of sensory input.
- *Environment* – Irrelevant. The environment of the network is a single operating system on a single computer.
- *Agility and Robustness* – Relevant. The users of the system will expect predictability and reliability.
- *Dynamic Structure and Scalability* – Relevant. Though not directly specified, it would be appropriate to create a system that would be easily modified to add additional sensor data as the system evolves.
- *Interaction* – Relevant. The system must provide an interface for the user to submit requests for trend data, as well as for viewing the current set of conditions.

After the partitioning of the evaluation considerations, the next step is to further develop the ranking by determining each evaluation considerations level of importance. The most important level of evaluation considerations consists of those that rate issues that are constraints of the software requirement

problem itself: *interaction*. As the system is more dependent on the real-time acquisition of data, *agility and robustness* will also be placed in this tier. The next level of ranking is for the evaluation considerations that are important issues for the success of the project, but the level of achievement may be negotiable. *Dynamic structure and scalability* has been assigned to this level. For this problem, *organizational business practices*, *methodology maturity*, and *cost of methodology and support tools* considerations were placed in a third tier, as a goal of the departments is to develop a well-defined and repeatable software engineering process.

The next step in refinement is determining the relative weighting of each evaluation consideration with regard to the next least important consideration. In this case, the evaluation considerations have been ordered alphabetically, so *interaction* is compared to *dynamic structure and scalability*, and *dynamic structure and scalability* is compared to *organizational business practices*. The tier of evaluation considerations that measure the constraints of the problem was determined by the decision maker to be twice as important as the lower rank. Likewise, the tier of negotiable considerations was determined to be one and a quarter times important as the lowest tier.

After determining the relative weighting, the normalized weights for each evaluation consideration must be determined following the method described in Section 2.3.1.3. For illustration, the algebraic calculations are shown below using the subscripts as defined in Figure 24.

$$w_{mat} = w_{org}$$

$$w_{cost} = w_{mat} = w_{org}$$

$$w_{dss} = 1.25 \cdot (w_{org})$$

$$w_{int} = 2 \cdot (w_{dss}) = 2 \cdot (1.25 \cdot w_{org})$$

$$w_{ar} = 1 \cdot (w_{cost}) = 1 \cdot (2 \cdot (1.25 \cdot w_{org}))$$

Setting the sum of the weights to 1,

$$1 = w_{org} + w_{cost} + w_{mat} + w_{dss} + w_{int} + w_{ar}$$

$= w_{org} + w_{org} + w_{org} + (1.25 ? w_{org}) + (2 ? (1.25 ? w_{org})) + (2 ? (1.25 ? w_{org}))$

$= w_{org} ? (1 + 1 + 1 + 1.25 + 2.5 + 2.5)$

$= w_{org} ? (9.25)$

$w_{org} = 1 / 9.25 = 0.108$

Hence,

$w_{mat} = 0.108$

$w_{cost} = 1 ? (0.108) = 0.108$

$w_{dss} = 1.25 ? (0.108) = 0.135$

$w_{int} = 2 ? (0.135) = 0.270$

$w_{ar} = 1 ? (0.232) = 0.270$

With the weights determined for the relevant evaluation considerations, the first step of the decision-making process is complete. The results of this step are summarized in Figure 36.

Rank	Evaluation Consideration	Relative Weight	Normalized Weight
1	Agility and Robustness	1	0.270
1	Interaction	2	0.270
2	Dynamic Structure and Scalability	1.25	0.135
3	Cost of Methodology & Support Tools	1	0.108
3	Methodology Maturity	1	0.108
3	Organizational Business Practices		0.108

Figure 39: Content Search Weighting Summary

4.4.2.2.2 Rating the Evaluation Considerations

The next step of the decision analysis is to score the evaluation considerations. The focus points are examined and scored for each of the evaluation considerations that have been determined to be relevant in the weighting step. This step is documented in Section 4.3.1. The single dimensional value functions for case 2 are summarized in Figure 40.

Evaluation Consideration	SDVF Fitness – MaSE	SDVF Fitness – Booch	SDVF Fitness – Yourdon
Agility and Robustness	0.417	0.250	0.167
Interaction	0.500	0.833	0.500
Dynamic Structure and Scalability	0.750	0.625	0
Cost of Methodology & Support Tools	0.936	0.591	0.378
Methodology Maturity	0.333	1.000	1.000
Organizational Business Practices	0.679	0.714	0.679

Figure 40: Single Dimensional Value Function Fitness Values-Case 2

4.4.2.2.3 Calculating the Multiobjective Value Functions

The final step is to calculate the multiobjective value function. To accomplish this step, the single dimensional value functions are multiplied by the appropriate weight and summed. For the MaSE alternative, the multiobjective value is determined by:

$$\begin{aligned}
\mathbf{V_{MaSE}(X)} = {} & w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{mat}v_{mat}(x_{mat}) + w_{ar}v_{ar}(x_{ar}) \\
& + w_{dss}v_{dss}(x_{dss}) + w_{int}v_{int}(x_{int}) \\
= {} & 0.108\, v_{cost}(1690) + 0.108\, v_{org}(19) + 0.108\, v_{mat}(4) + 0.27\, v_{ar}(5) \\
& + 0.135\, v_{dss}(6) + 0.27\, v_{int}(6) \\
= {} & 0.101 + 0.073 + 0.036 + 0.113 + 0.101 + 0.135 \\
= {} & \mathbf{0.559}
\end{aligned}$$

The Booch alternative is calculated to be:

$$\begin{aligned}
\mathbf{V_{Booch}(X)} = {} & w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{mat}v_{mat}(x_{mat}) + w_{ar}v_{ar}(x_{ar}) \\
& + w_{dss}v_{dss}(x_{dss}) + w_{int}v_{int}(x_{int}) \\
= {} & 0.108\, v_{cost}(5454) + 0.108\, v_{org}(20) + 0.108\, v_{mat}(12) + 0.27\, v_{ar}(3) \\
& + 0.135\, v_{dss}(5) + 0.27\, v_{int}(10) \\
= {} & 0.064 + 0.077 + 0.108 + 0.068 + 0.084 + 0.225 \\
= {} & \mathbf{0.626}
\end{aligned}$$

The Yourdon alternative is calculated to be:

$$\mathbf{V_{Yourdon}(X)} = w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{mat}v_{mat}(x_{mat}) + w_{ar}v_{ar}(x_{ar})$$

$$+ w_{dss}v_{dss}(x_{dss}) + w_{int}v_{int}(x_{int})$$

$$= 0.108\, v_{cost}(6610) + 0.108\, v_{org}(19) + 0.108\, v_{mat}(12) + 0.27\, v_{ar}(2)$$

$$+ 0.135\, v_{dss}(0) + 0.27\, v_{int}(6)$$

$$= 0.041 + 0.073 + 0.108 + 0.045 + 0 + 0.135$$

$$= \mathbf{0.402}$$

4.4.2.2.4 Determining the Best Alternative

Based on the above calculations and assumptions, the decision analysis tool has determined the Booch methodology to be the better of the three approaches. The differences in scores, in this case, are greater than in Case 1, but there is still important information to be gained by a sensitivity analysis. Again, Section 4.5 describes the results of the sensitivity analysis on this case study.

4.4.2.3 Case 3: Cryptanalysis

The third case study evaluated is an intelligent system. The cryptanalysis system was used as a case study by Booch to demonstrate the object-oriented architecture's applicability in creating intelligent problem solvers. The system specification is in Figure 41. For this system, a brute force approach would be useless as there are 26! possible combinations if only one case of the alphabet is used [2]. Adding the other case, numbers, and other symbols, ninety-nine symbols could be used!

The architecture suggested by Booch is a "Blackboard" architecture. The blackboard is an area where autonomous knowledge sources can determine the state of the cryptogram and apply its specialized knowledge. The blackboard keeps track of modifications and can institute rollbacks when necessary to allow backtracking. This approach is compared to the possible use of an agent-oriented approach.

4.4.2.3.1 Weighting the Evaluation Considerations

The first step in weighting the evaluation considerations is to determine which of the evaluation considerations are relevant to the particular problem. Below are each of the categories and the analysis decision made as to whether or not the consideration is relevant.

- *Cost of Acquiring Methodology and Support Tools* – Relevant: the approach to this problem is that the software engineer is a part of an organization that is looking to adopt the methodology and supporting tools.
- *Organizational Business Practices* – Relevant: although the software engineer is the only employee in the fledgling department, the engineer does have the responsibility of providing project updates to other interested parties outside of the department.
- *Methodology Maturity* – Relevant: the decision to change to a new methodology will require some degree of evidence that it will produce quality software.
- *Integration of Reusable Components* – Irrelevant: a library of reusable components is not available to the software engineer.
- *Legacy System Integration* – Irrelevant: the system is not required to incorporate any existing software systems.
- *Distribution* – Relevant: translation components should be able to be placed on systems throughout the network that best leverage the computing power for better algorithm performance.
- *Environment* – Relevant: the environment of the network is a mixture of Sun Workstations running Solaris OS and Personal Computers running Windows NT. Translation components will be placed on systems that have the appropriate level of computing power.

- *Agility and Robustness* – Relevant: the users of the system will expect predictability and reliability.
- *Dynamic Structure and Scalability* – Relevant: the system needs to be able to incorporate translation components as they become available.
- *Interaction* – Relevant: at the very least, the system must be able to take an input file for decryption, and provide the user with information regarding the potential translations.

Now that the evaluation considerations have been partitioned, the ranking must be refined by determining the relative importance of each evaluation consideration. The most important level of evaluation considerations consists of those that rate issues that are constraints of the software requirement problem itself: *distribution*, *environment*, and *dynamic structure and scalability*. The next level of ranking is for the evaluation considerations that are important issues for the success of the project, but the level of achievement may be negotiable. *Agility and robustness*, *interaction*, *organizational business practices*, *methodology maturity*, and *cost of methodology and support tools* have been assigned to this lower level.

The next step in refinement is determining the relative weighting of each evaluation consideration with regard to the next least important consideration. In this case, the evaluation considerations have again been ordered alphabetically, so *interaction* is compared to *agility and robustness*, and *dynamic structure and scalability* is compared to *organizational business practices*. The tier of evaluation considerations that measure the constraints of the problem was determined by the decision maker to be twice as important as the lower rank. Likewise, the tier of negotiable considerations was determined to be one and a quarter times important as the lowest tier.

Cryptanalysis Requirements

Cryptography "embraces methods for rendering data unintelligible to unauthorized parties." Using cryptographic algorithms, messages (plaintext) may be transformed into cryptograms (ciphertext) and back again.

One of the most basic kinds of cryptographic algorithms, employed since the time of the Romans, is called a substitution cipher. With this cipher, every letter of the plaintext alphabet is mapped to a different letter. For example, we might shift every letter to its successor: ***A*** *becomes* ***B***, ***B*** *becomes* ***C***, ***Z*** *wraps around to become* ***A***, *and so on. Thus, the plaintext*

CLOS is an object-oriented programming language

may be enciphered to the cryptogram

DMPT jt bo pckfdu-psjfoufe qsphsbnnjoh mbohvbhf

Most often, the substitution of the letters is jumbled. For example, ***A*** *becomes* ***G***, ***B*** *becomes* ***J***, *and so on. As an example, consider the following cryptogram:*

PDG TBCER CQ TCK AL S NGELCH QZBBR SBAJG

Hint: the letter ***C*** *represents the plaintext letter* ***O***.

It is a vastly simplifying assumption to know that only a substitution cipher was employed to encode a plaintext message; nevertheless, deciphering the resulting cryptogram is not an algorithmically trivial task. Deciphering sometimes requires trial and error, wherein we make assumptions about a particular substitution and then evaluate their implications. For example, we may start with the one- and two-letter words in the cryptogram and hypothesize that they stand for common words such as ***I*** *and* ***a***, *or* ***it***, ***in***, ***is***, ***of***, ***or***, *and* ***on***. *By substituting the other occurrences of these ciphered letters, we may find hints for deciphering other words. For instance, if there is a three-letter word that starts with an* ***o***, *the word might reasonably be* ***one***, ***our***, *or* ***off***.

We can also use our knowledge of spelling and grammar to attack a substitution cipher. For example, an occurrence of double letters is not likely to represent the sequence ***qq***. *Similarly, we might try to expand a word ending with the letter* ***g*** *to the suffix* ***ing***. *At a higher level of abstraction, we might assume that the sequence of words* ***it is*** *is more likely to occur than the sequence* ***if is***. *Also, we might assume that the structure of a sentence typically includes a noun and a verb. Thus, if our analysis has identified a verb but no actor or agent, we might start a search for adjectives and nouns.*

Sometimes we may have to backtrack. For example, we might have assumed that a certain two-letter word was ***or***, *but if the substitution for the letter* ***r*** *causes contradictions or blind alleys in other words, then we might have to try the word* ***of*** *or* ***on*** *instead, and consequently undo other assumptions we had based upon this earlier substitution.*

This leads us to the requirement of our problem: devise a system that, given a cryptogram, transforms it back to its original plaintext, assuming that only a simple substitution cipher was employed.

Figure 41: Cryptanalysis Requirements [2]

With the relative weighting determined, the next step is to determine the normalized weights for each evaluation consideration following the method described in Section 2.3.1.3. For illustration, the algebraic calculations are shown below using the subscripts as defined in Figure 24.

$w_{cost} = w_{org}$

$w_{int} = 1 ? w_{cost}$

$w_{ar} = 1 ? w_{int} = 1 ? (w_{org})$

$w_{mat} = 1 ? (w_{ar}) = 1 ? (1 ? w_{org})$

$w_{env} = 2 ? (w_{mat}) = 2 ? (1 ? (1 ? w_{org}))$

$w_{dss} = 1 ? (w_{env}) = 1 ? (2 ? (1 ? (1? w_{org})))$

$w_{dis} = 1 ? (w_{dss}) = 1 ? (1 ? (2 ? (1 ? (1? w_{org}))))$

Setting the sum of the weights to 1,

$$1 = w_{cost} + w_{org} + w_{mat} + w_{dss} + w_{ar} + w_{int} + w_{env} + w_{dis}$$

$$= w_{org} + w_{org} + w_{org} + w_{org} + (2 ? w_{org}) + (2 ? w_{org}) + (2 ? w_{org}) + (2 ? w_{org})$$

$$= w_{org} ? (1 + 1 + 1 + 1 + 1 + 2 + 2 + 2)$$

$$= w_{org} ? (11)$$

$w_{org} = 1 / 11 = 0.091$

Hence,

$w_{cost} = w_{org} = 0.091$

$w_{int} = w_{cost} = 0.091$

$w_{ar} = w_{int} = 0.091$

$w_{mat} = w_{ar} = 0.091$

$w_{env} = 2 ? (0.091) = 0.182$

$w_{dss} = w_{env} = 0.182$

$w_{dis} = w_{dss} = 0.182$

With the weights determined for the relevant evaluation considerations, the first step of the decision-making process is complete. The results of this step are summarized in Figure 36.

Rank	Evaluation Consideration	Relative Weight	Normalized Weight
1	Distribution	1	0.182
1	Dynamic Structure and Scalability	1	0.182
1	Environment	2	0.182
2	Cost of Methodology & Support Tools	1	0.091
2	Methodology Maturity	1	0.091
2	Agility and Robustness	1	0.091
2	Interaction	1	0.091
2	Organizational Business Practices		0.091

Figure 42: Content Search Weighting Summary

4.4.2.3.2 Rating the Evaluation Considerations

The next step of the decision analysis is to score the evaluation considerations. The focus points are examined and scored for each of the evaluation considerations that have been determined to be relevant in the weighting step. This step is documented in Section 4.3.1. The single dimensional value functions for case 3 are summarized in Figure 45.

Evaluation Consideration	SDVF Fitness – MaSE	SDVF Fitness – Booch	SDVF Fitness – Yourdon
Distribution	0.750	0.500	0.083
Dynamic Structure and Scalability	0.750	0.625	0
Environment	0.833	0.833	0.333
Cost of Methodology & Support Tools	0.936	0.591	0.378
Methodology Maturity	0.333	1.000	1.000
Agility and Robustness	0.417	0.250	0.167
Interaction	0.500	0.833	0.500
Organizational Business Practices	0.679	0.714	0.679

Figure 43: Single Dimensional Value Function Fitness Values-Case 3

4.4.2.3.3 Calculating the Multiobjective Value Functions

The final step is to calculate the multiobjective value function. To accomplish this step, the single dimensional value functions are multiplied by the appropriate weight and summed. For the MaSE alternative, the multiobjective value is determined by:

$$\mathbf{V_{MaSE}(X)} = w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{mat}v_{mat}(x_{mat}) + w_{dis}v_{dis}(x_{dis}) + w_{env}v_{env}(x_{env}) + w_{ar}v_{ar}(x_{ar}) + w_{dss}v_{dss}(x_{dss}) + w_{int}v_{int}(x_{int})$$

$$= 0.091\, v_{cost}(1690) + 0.091\, v_{org}(19) + 0.091 v_{mat}(4) + 0.182\, v_{dis}(9) + 0.182 v_{env}(10) + 0.091\, v_{ar}(5) + 0.182\, v_{dss}(6) + 0.091\, v_{int}(6)$$

$$= 0.085 + 0.062 + 0.030 + 0.137 + 0.152 + 0.038 + 0.137 + 0.046$$

$$= \mathbf{0.685}$$

The Booch alternative is calculated to be:

$$\mathbf{V_{Booch}(X)} = w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{dis}v_{dis}(x_{dis}) + w_{env}v_{env}(x_{env}) + w_{ar}v_{ar}(x_{ar}) + w_{dss}v_{dss}(x_{dss}) + w_{int}v_{int}(x_{int})$$

$$= 0.091\, v_{cost}(5454) + 0.091\, v_{org}(20) + 0.091 v_{mat}(12) + 0.182\, v_{dis}(6) + 0.182 v_{env}(10) + 0.091\, v_{ar}(3) + 0.182\, v_{dss}(5) + 0.091\, v_{int}(10)$$

$$= 0.054 + 0.065 + 0.091 + 0.091 + 0.152 + 0.023 + 0.114 + 0.076$$

$$= \mathbf{0.665}$$

The Yourdon alternative is calculated to be:

$$\mathbf{V_{Yourdon}(X)} = w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{dis}v_{dis}(x_{dis}) + w_{env}v_{env}(x_{env}) + w_{ar}v_{ar}(x_{ar}) + w_{dss}v_{dss}(x_{dss}) + w_{int}v_{int}(x_{int})$$

$$= 0.091\, v_{cost}(6610) + 0.091\, v_{org}(19) + 0.091 v_{mat}(12) + 0.182\, v_{dis}(1) + 0.182 v_{env}(4) + 0.091\, v_{ar}(2) + 0.182\, v_{dss}(0) + 0.091\, v_{int}(6)$$

$$= 0.034 + 0.062 + 0.091 + 0.015 + 0.061 + 0.015 + 0 + 0.046$$

$$= \mathbf{0.324}$$

4.4.2.3.4 Determining the Best Alternative

Based on the above calculations and assumptions, the decision analysis tool has determined the MaSE methodology to score higher than the other two approaches. Considering the nature of this problem, a multiagent approach makes sense. In fact, Booch's description of the problem is suggestive of software agents [2]. Again, Section 4.5 contains the sensitivity analysis on this case study.

4.4.2.4 Case 4: Inventory Tracking System

The fourth case study examined is another information management system. The Inventory Tracking System provides the regional warehouse with control of inventory management, while it provides a central headquarters with reporting capability. The requirements for the Inventory Tracking System are in Figure 44.

Inventory Tracking System Requirements

As part of its expansion into several new and specialized markets, a mail order catalog company has decided to establish a number of relatively autonomous regional warehouses. Each such warehouse retains local responsibility for inventory management and order processing. To target niche markets efficiently, each warehouse is tasked with maintaining inventory that is best suited to the local market. The specific product line that each warehouse manages may differ from region to region; furthermore, the product line managed by any one region tends to be updated almost yearly to keep up with changing customer tastes. For reasons of economies of scale, the parent company desires to have a common inventory- and order- tracking system across all its warehouses.

The key functions of this system include:

- Tracking inventory as it enters the warehouse, shipped from a variety of suppliers.

- Tracking orders as they are received from a central but remote telemarketing organization; orders may also be received by mail, and are processed locally.

- Generating packing slips, used to direct warehouse personnel in assembling and then shipping an order.

- Generating invoices and tracking accounts receivable.

- Generating supply requests and tracking accounts payable.

In addition to automating much of the warehouse's daily workflow, the system must provide a general and open-ended reporting facility, so that the management team can track sales trends, identify valued and problem customers and suppliers, and carry out special promotional programs.

Figure 44: Inventory Tracking System Requirements [2]

One of the significant differences between this system and the Content Search System is the multiple levels of distribution. Functionality of the inventory management is distributed among many users within the local system users. Data gathering is distributed amongst the many regional warehouses. Another difference between this information system and the other discussed is the economy of scale issue. The organization is attempting to reduce costs by developing a single system, which negates any impact on the *environment* consideration.

4.4.2.4.1 Weighting the Evaluation Considerations

Again, the first step in weighting the evaluation considerations is to determine which of the evaluation considerations are relevant to the particular problem. Below are each of the categories and the analysis decision made as to whether or not the consideration is relevant.

- *Cost of Acquiring Methodology and Support Tools* – Relevant: the approach to this problem is that the software engineer is a part of an organization that is looking to adopt the methodology and supporting tools.
- *Organizational Business Practices* – Relevant: although the software engineer is the only employee in the fledgling department, the engineer does have the responsibility of providing project updates to other interested parties outside of the department.
- *Methodology Maturity* – Relevant: the decision to change to a new methodology will require some degree of evidence that it will produce quality software.
- *Integration of Reusable Components* – Irrelevant: a library of reusable components is not available to the software engineer.
- *Legacy System Integration* – Relevant: the system is incorporating existing software systems for auxiliary functionality.
- *Distribution* – Relevant: the users of the system will require access from different nodes on the network. Likewise, the data that the users will require is stored on many hard drives throughout the network.
- *Environment* – Irrelevant: the environment of the network is homogeneous.
- *Agility and Robustness* – Relevant: the users of the system will expect predictability and reliability.

- *Dynamic Structure and Scalability* – Irrelevant: the opportunity for incorporating additional nodes in this system is not being considered.
- *Interaction* – Relevant: the system must provide an interface for the user to manage inventory as well as retrieve reports.

Following this basic separation of the evaluation considerations, the next step is to develop the ranking by determining each evaluation consideration's level of importance. The most important level of evaluation considerations consists of those that rate issues that are constraints of the software requirement problem itself: *cost of methodology and support tools*, *distribution*, and *interaction*. The next level of ranking is for the evaluation considerations that are important issues for the success of the project, but the level of achievement may be negotiable. A*gility and robustness* and *legacy system integration* have been assigned to this level. For this problem, *organizational business practices* and *methodology maturity* considerations were placed in a third tier.

The next step in refinement is determining the relative weighting of each evaluation consideration with regard to the next least important consideration. In this case, as previously, the evaluation considerations have been ordered alphabetically, so *interaction* is compared to *agility and robustness*, and *legacy system integration* is compared to *organizational business practices*. The tier of evaluation considerations that measure the constraints of the problem was determined by the decision maker to be twice as important as the lower rank. Likewise, the tier of negotiable considerations was determined to be one and a half times important as the lowest tier.

With the relative weighting determined, the next step is to determine the normalized weights for each evaluation consideration following the method described in Section 2.3.1.3. For illustration, the algebraic calculations are shown below using the subscripts as defined in Figure 24.

$w_{org} = w_{mat}$

$w_{leg} = 1.5 ? (w_{mat})$

$w_{ar} = 1 ? (w_{leg}) = 1 ? (1.5 ? w_{mat})$

$w_{int} = 2 ? (w_{ar}) = 2 ? (1 ? (1.5 ? w_{mat}))$

$w_{dis} = 1 ? (w_{int}) = 1 ? (2 ? (1 ? (1.25 ? w_{mat})))$

$w_{cost} = 1 ? (w_{dis}) = 1 ? (1 ? (2 ? (1 ? (1.25 ? w_{mat}))))$

Setting the sum of the weights to 1,

$$1 = w_{org} + w_{mat} + w_{leg} + w_{ar} + w_{int} + w_{dis} + w_{cost}$$

$$= w_{mat} + w_{mat} + (1.5 ? w_{mat}) + (1.5 ? w_{mat}) + (2 ? (1 ? (1.5 ? w_{mat})))$$
$$+ (2 ? (1 ? (1.5 ? w_{mat}))) + (2 ? (1 ? (1.5 ? w_{mat})))$$

$$= w_{mat} ? (1 + 1 + 1.5 + 1. 5 + 3 + 3 + 3)$$

$$= w_{mat} ? (14)$$

$w_{mat} = 1 / 14 = 0.071$

Hence,

$w_{org} = 0.071$

$w_{leg} = 1.5 ? (0.071) = 0.107$

$w_{ar} = 1 ? (0.107) = 0.107$

$w_{int} = 2 ? (0.107) = 0.214$

$w_{dis} = 1 ? (0.214) = 0.214$

$w_{cost} = 1 ? (0.214) = 0.214$

With the weights determined for the relevant evaluation considerations, the first step of the decision-making process is complete. The results of this step are summarized in Figure 36.

Rank	Evaluation Consideration	Relative Weight	Normalized Weight
1	Cost of Methodology & Support Tools	1	0.214
1	Distribution	1	0.214
1	Interaction	2	0.214
2	Agility and Robustness	1	0.107
2	Legacy System Integration	1.5	0.107
3	Organizational Business Practices	1	0.071
3	Methodology Maturity		0.071

Figure 45: Inventory Tracking System Weighting Summary

4.4.2.4.2 Rating the Evaluation Considerations

The next step of the decision analysis is to score the evaluation considerations. The focus points are examined and scored for each of the evaluation considerations that have been determined to be relevant in the weighting step. This step is documented in Section 4.3.1. The single dimensional value functions for case 4 are summarized in Figure 46.

Evaluation Consideration	SDVF Fitness – MaSE	SDVF Fitness – Booch	SDVF Fitness – Yourdon
Cost of Methodology & Support Tools	0.936	0.591	0.378
Distribution	0.750	0.500	0.083
Interaction	0.500	0.833	0.500
Agility and Robustness	0.417	0.250	0.167
Legacy System Integration	0.417	0.333	0.083
Organizational Business Practices	0.679	0.714	0.679
Methodology Maturity	0.333	1.000	1.000

Figure 46: Single Dimensional Value Function Fitness Values-Case 4

4.4.2.4.3 Calculating the Multiobjective Value Functions

The final step is to calculate the multiobjective value function. To accomplish this step, the single dimensional value functions are multiplied by the appropriate weight and summed. For the MaSE alternative, the multiobjective value is determined by:

$$\begin{aligned} \mathbf{V_{MaSE}(X)} = {} & w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{mat}v_{mat}(xmat) + w_{leg}v_{leg}(x_{leg}) \\ & + w_{dis}v_{dis}(x_{dis}) + w_{ar}v_{ar}(x_{ar}) + w_{int}v_{int}(x_{int}) \\ = {} & 0.214\, v_{cost}(1690) + 0.071\, v_{org}(19) + 0.071 v_{mat}(4) + 0.107\, v_{leg}(5) \\ & + 0.214\, v_{dis}(9) + 0.107\, v_{ar}(5) + 0.214\, v_{int}(6) \\ = {} & 0.200 + 0.048 + 0.024 + 0.045 + 0.161 + 0.045 + 0.107 \\ = {} & \mathbf{0.629} \end{aligned}$$

The Booch alternative is calculated to be:

$$\begin{aligned} \mathbf{V_{Booch}(X)} = {} & w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{mat}v_{mat}(x_{mat}) + w_{leg}\, v_{leg}(x_{leg}) \\ & + w_{dis}v_{dis}(x_{dis}) + w_{ar}v_{ar}(x_{ar}) + w_{int}v_{int}(x_{int}) \end{aligned}$$

$$= 0.214\, v_{cost}(5454) + 0.071\, v_{org}(20) + 0.071 v_{mat}(12) + 0.107\, v_{leg}(4) + 0.214\, v_{dis}(6) + 0.107\, v_{ar}(3) + 0.214\, v_{int}(10)$$

$$= 0.127 + 0.051 + 0.071 + 0.036 + 0.107 + 0.027 + 0.178$$

$$= \mathbf{0.596}$$

The Yourdon alternative is calculated to be:

$$\mathbf{V_{Yourdon}(X)} = w_{cost}v_{cost}(x_{cost}) + w_{org}v_{org}(x_{org}) + w_{mat}v_{mat}(x_{mat}) + w_{leg}\, v_{leg}(x_{leg}) + w_{dis}v_{dis}(x_{dis}) + w_{ar}v_{ar}(x_{ar}) + w_{int}v_{int}(x_{int})$$

$$= 0.214\, v_{cost}(6610) + 0.071\, v_{org}(19) + 0.071 v_{mat}(12) + 0.107\, v_{leg}(1) + 0.214\, v_{dis}(1) + 0.107\, v_{ar}(2) + 0.214\, v_{int}(6)$$

$$= 0.081 + 0.048 + 0.071 + 0.009 + 0.018 + 0.018 + 0.107$$

$$= \mathbf{0.352}$$

4.4.2.4.4 Determining the Best Alternative

Based on the above calculations and assumptions, the decision analysis tool has determined the MaSE methodology to score higher than the other two approaches. The problem domain of information systems, such as this one, is often used as an example of multiagent applications [36]. The difference in the score of the MaSE and Booch methodologies is 0.033. It is reasonable to conclude that both approaches are suitable, as Booch used the case study as an example for his textbook [2]. For more insight to the decision, the sensitivity analysis is available in Section 4.5.

4.5 Analysis of the Assumptions

The two factors that determine the value of the multiobjective are the weights assigned to each evaluation consideration and the score the alternatives receive for each evaluation consideration. Though the ratings of each of the focus points are subjective, each rating was based on the experience of the assessor and assumed to be accurate. The weights, on the other hand, are strictly based on the opinion that one evaluation consideration is some quantity more than another.

In order to give the assessor a different perspective on the decision, sensitivity analyses have been performed on each of the evaluation considerations relevant to the software requirement. These analyses are examined for each of the case studies. These sensitivity analyses will focus on the areas where a slight change in an evaluation consideration's weight significantly changes the fitness score. Before beginning the analysis, Figure 47 summarizes the fitness scores the methodologies rated.

Case Study	MaSE MFV	Booch MFV	Yourdon MFV
Content Search System	**0.689**	0.668	0.353
Weather Monitoring Station	0.559	**0.626**	0.402
Cryptanalysis	**0.685**	0.665	0.324
Inventory Tracking System	**0.629**	0.596	0.352

Figure 47: Case Study Multiobjective Fitness Values

The first analysis looks at the initial selection of weights. The process of selecting the relative weights is designed to establish an intuitive relationship between evaluation considerations. The weights are then normalized, generating the appropriate percentages of the whole. Figure 11 shows the results of the survey discussed in Section 3.2 with regard to partitioning the weighting of the management and technical evaluation considerations. The process used to determine the normalized weights produced results within the range of suggestions. For each case study, the composite percentages of the management and technical considerations are summarized in Figure 48.

Case Study	Management Considerations	Technical Considerations
Content Search System	31%	69%
Weather Monitoring Station	32%	68%
Cryptanalysis	27%	73%
Inventory Tracking System	36%	64%

Figure 48: Composite Percentages of Management and Technical Considerations

Using the Data Analyzer tool that has been developed to work with the output of the decision analysis tool implementation a full sensitivity analysis can be performed for all of evaluation considerations. The analysis focuses on the most sensitive of the considerations with regard to the original normalized weight in order to highlight the different possible results of the decision analysis tool. This is done by evaluating the fitness of each methodology over a range of weights for a particular methodology. For the case studies evaluated here, the entire range of possible weight, 0 to 1, is calculated. In order to

ensure that the total normalized weight remains equal to 1, the considerations that are not being tested are calculated to be proportional to the total weight minus the weight of the consideration being tested. This method ensures that only the evaluation consideration being tested effects the rating. The decision is considered sensitive if a small change to the weight produces a change. Recommendations for a defining a "small change" are a change of 5% or up to 7% [16, 18]. Additionally, a recommendation for a decision is less definitive when the ratio of sensitive considerations to the total number of considerations exceeds 33% [18].

Detecting sensitivity relies on the identification of critical points. It is at this point that the weight for the particular consideration changes the decision analysis tool's preference. An example of the a sensitivity analysis chart, the sensitivity analysis on *Methodology Maturity* for Case Study 2, is in Figure 49. The entire set of sensitivity analysis charts can be found in Appendix C.

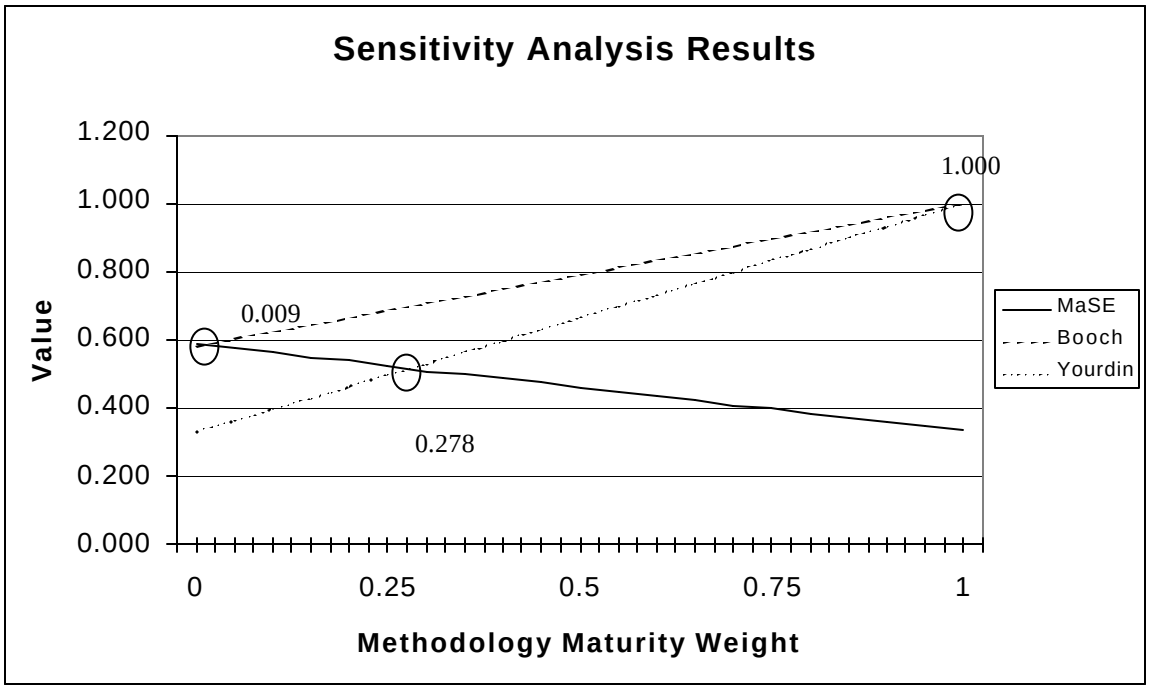


Figure 49: Sensitivity Analysis Example

The sensitivity analysis in Figure 47 is annotated to point out the critical points in this particular analysis. For example, this figure has three critical points. The first is when the weight for *Methodology*

Maturity is 0.009. When the weight is within the range 0 to 0.009, the MaSE methodology has the highest multiobjective fitness value. The second critical point, at 0.278, is the weight that the Yourdin methodology rates begins to rate higher than MaSE, however, it is still less than the Booch methodology. The third critical point, 1.000, is where the Booch and Yourdin methodologies intersect. It should be noted, however, that if the weight of *Methodology Maturity* is set to 1.000, no other factors are playing into the decision.

For all of the case studies, a set of evaluation considerations existed that any change in the weight of the particular consideration would yield no different answer. Additionally, some of the considerations would change the preference but only at a significant change. The critical points and original weights for each of the case studies are shown in Figure 50. In the case that a consideration was not used in a particular case study, "N/A" is entered under the original value and the critical point. For the considerations where there was not a critical point, "-" is entered under the critical point column. Additionally, the amount of the change needed to alter the original weight to the weight needed to switch preferences. Changes less than and equal to 7% are highlighted.

The least sensitive decisions were made when considering Case Studies 2 and 4. Based on the requirement statements for these two problems, the Booch and MaSE methodologies were selected respectively. Case Study 1 and 3 each had a high degree of sensitivity. The decision analysis tool recommended MaSE for both of these solutions, but the sensitivity analysis indicates that if the user's weighting preferences were slightly different, the Booch methodology would be the preference. In all likelihood, either methodology would satisfy the user's needs. The decisions for Case Studies 2 and 4 were much more resilient to change. In these cases, the recommended methodologies would provide the user with the best support.

For the three case studies that were taken from Booch, the decision made at the time was that his object-oriented methodology was better suited for handling complex software systems than the traditional functional approach. This decision is validated in that the functional approach to software engineering, Yourdin's methodology, scored substantially lower for this set of considerations. Additionally, the

development of agent-oriented methodologies, like MaSE, provides techniques that offer even more support for many of the aspects of software requirements that make software complex. Section 4.6 provides further validation of the decision.

4.6 Decision Validation

In order to validate the decision analysis tool's ability to report good results, Case Study 1 from Section 4.4.2 was implemented using both the MaSE and Booch methodologies. When analyzed with the decision analysis tool, the MaSE methodology's multiobjective fitness value was calculated to be 68.9% and the Booch methodology's was 66.8%. The sensitivity analyses on the weights for this case study revealed three evaluation considerations in which critical points were within 7% of the original weights. The purpose of developing the software requirement was to validate the hypothesis that the MaSE methodology is a "better" choice for solving the Content Search problem.

During the development of the product, the set of metrics described in Figure 8 were collected. The data collected is presented in Figure 51. Additionally, the products of the modeling are in Appendix E. From the data collected, a few items stand out. First, it should be noted that unit testing was conducted during the implementation phase, and several hours were spent implementing and testing the remote method invocation used in the Booch approach. Next, the amount of code produced using the MaSE approach includes code generated by the methodology's CASE tool. This code implements a specific code framework that has been developed to demonstrate the methodology [39]. This framework, called AgentMOM, contributes to the cyclomatic complexity of the implementation of the agent conversation state diagrams. A feature of the CASE tool is its ability to test the state diagrams prior to being transformed for unvisited states and deadlock, so aspects of the cyclomatic complexity are verified prior to code generation.

Evaluation Consideration	MaSE – Booch Critical Point	MaSE – Yourdin Critical Point	Booch – Yourdin Critical Point	Original Weight	Change (+/-)
CASE 1					
Cost	0.117	-	-	0.172	**0.056**
Org	0.419	1.000	-	0.069	0.350
Leg	N/A	N/A	N/A	0	N/A
Dis	0.094	-	-	0.172	0.078
Env	1.000	-	-	0.172	0.828
AR	-	-	-	0.086	-
DSS	-	-	-	0.086	-
Int	0.223	1.000	-	0.172	**0.051**
Mat	0.098	0.382	1.000	0.069	**0.029**
CASE 2					
Cost	0.253	-	-	0.108	0.145
Org	-	1.000	-	0.108	0.892
Leg	N/A	N/A	N/A	0	N/A
Dis	N/A	N/A	N/A	0	N/A
Env	N/A	N/A	N/A	0	N/A
AR	0.479	-	-	0.270	0.209
DSS	0.437	-	-	0.135	0.302
Int	0.087	1.000	-	0.270	0.083
Mat	0.009	0.278	1.000	0.108	0.099
CASE 3					
Cost	0.033	-	-	0.091	**0.058**
Org	0.422	1.000	-	0.091	0.331
Leg	N/A	N/A	N/A	N/A	N/A
Dis	0.109	-	-	0.182	0.073
Env	1.000	-	-	0.182	0.818
AR	-	-	-	0.091	-
DSS	0.022	-	-	0.182	0.160
Int	0.143	1.000	-	0.091	**0.052**
Mat	0.118	0.410	1.000	0.091	**0.027**
CASE 4					
Cost	0.132	-	-	0.214	0.082
Org	0.515	1.000	-	0.071	0.444
Leg	-	-	-	0.107	N/A
Dis	0.096	-	-	0.214	0.118
Env	N/A	N/A	N/A	N/A	N/A
AR	-	-	-	0.107	-
DSS	N/A	N/A	N/A	N/A	N/A
Int	0.285	1.000	-	0.214	0.071
Mat	0.115	0.344	1.000	0.071	**0.044**

Figure 50: Weights and Critical Point Summary

Metric	MaSE Approach	Booch Approach
Modeling Effort – Analysis	4.83 labor hours	4.53 labor hours
Modeling Effort – Design	2.17 labor hours	4.08 labor hours
Modeling Effort – Implementation	8.17 labor hours	11.75 labor hours
Size – SLOC	1252	638
Size – Classes	20	11
Cyclomatic Complexity	74	6
Size/Effort Ratio	153.2 SLOC/labor hour	54.3 SLOC/labor hour

Figure 51: Content Search Development Metrics

From this set of data, it is evident that the production of code is greatly enhanced by using MaSE and its CASE tool. Though the complexity of the code is more significant than the code generated for the Booch methodology, the design models can maintain the software in the MaSE methodology. The Booch methodology and its CASE tool did not provide a direct mapping into code.

In addition to collecting the metrics on the two implementations, the models were presented to a group of graduate students in a software engineering program. Each student was presented one set of models, either MaSE or Booch, and a questionnaire. The questionnaire, models, and a summary of student responses are in Appendix E. The focus of the questionnaire was to determine the students' ability to describe certain technical issues that the methodology was required to address for this case study. Additionally, the questionnaire measured the users ability to understand the models developed by using the two methodologies. In order to generate the packets of models, models and data dictionaries were captured from the respective CASE tools. To compare the packages, the MaSE package had twenty-two models and a three-page data dictionary. The Booch package, on the other hand, had six models and a twelve-page data dictionary.

The first set of questions the respondents answered was to their familiarity with methodologies they were evaluating. Eight of the nine students reviewing the Booch models indicated that they were familiar with the methodology, and five of those indicated that they had developed systems using the methodology in the past. Only six of ten students indicated that they were familiar with the MaSE methodology, and of those, only five students had actually used the methodology for system development. These results were not unexpected as MaSE is a recently defined agent-oriented methodology and Booch's

object-oriented methodology is much more mature. Asked to identify other methodologies with which they are familiar, the students indicated object-oriented techniques, functional decomposition techniques, and ad hoc methods for developing software.

The next question was a general question about the respondents' confidence that they understood the models. Each was asked to rate his confidence on a scale of zero to four, with four indicating the greatest confidence in understanding the system. On the average, the understanding rating for the students evaluating the MaSE methodology was 3.2. The rating was 3.125 for the students evaluating the Booch methodology. Seven of the eight students reviewing the Booch methodology were able to identify the correct statement of description for the system. Only two of the ten students reviewing the MaSE methodology were able to select the correct statement; the other eight students selected the "nearly" correct answer.

The next set of questions looked at a number of details in the models, including the identification of legacy systems, reusable components, the network environment, and interface issues. The students reviewing the Booch methodology were divided equally with regard to identifying a legacy system. Based on the fact that there was not a legacy system incorporated in this system and the responses as to what the legacy system could possibly be—answers given included the *File System* and the *File Manager*—the naming convention was likely the reason for the misidentification. Only one student misidentified the legacy system in the set of MaSE models.

Determining the network environment was the intention of several questions. Identifying the configuration of the network the system was being designed for is important information that needs to be communicated to the developers. These questions measured the respondents' ability to discover this environment information. The group evaluating the MaSE example was able to more completely identify the hardware and software system components in the models. With regard to the user interface, both groups were able to identify the input and output of the systems as well as the options.

Based on the scoring included next to each question on the questionnaire in Appendix E, the average scores are 25.5 for the MaSE group and 25.4 for the Booch group. Furthermore, by considering the results for the students who were familiar and experienced with the respective methodology, the average score for the MaSE group was 27.2 and for the Booch group was 25.1.

With regard to the data collected during the development of the Content Search system and the questionnaire, there are positives and negatives with each approach. Experienced developers have demonstrated that the requirements of the system are best expressed and interpreted in the representations of the MaSE methodology. This conclusion agrees with the results of the decision analysis tool for the Content Search system as described in Section 4.4.2.

4.7 Summary

This chapter demonstrates how to apply the decision analysis tool by working through several software requirement problems. Completing the four distinct steps of the decision analysis process ends with assigning multiobjective values to the alternatives being compared and selecting an alternative. In addition to completing the analysis process on the four problems, different techniques were demonstrated for analyzing the sensitivity of the assumptions made in the first step of the process. By looking at the different possible values assigned to the assumptions, conditions can be determined as to which methodology is superior to another.

Chapter V provides a summary of the research by drawing conclusions on the data collected throughout this chapter. Additionally, future areas of research are suggested as they apply specifically to this research and the overall problem of selecting an appropriate methodology.

V. Conclusions and Future Work

5.1 Conclusions

One of the major questions currently being discussed in the field of multiagent systems is, when are multiagent systems appropriate to use? Throughout the current literature, statements such as, "if data existed for real industrial strength applications, quantitatively based decisions could be made," can be found lamenting the problem. As that data does not exist, researchers are left to defend their decisions to use a multiagent system approach with qualitative arguments that it is the next step in software evolution and theoretical hypotheses that the MAS approach will produce higher quality software.

Researchers continue to attempt to tackle this question by providing intellectual—qualitative—arguments, like Wooldridge and Jennings [14, 15, 41] or by developing "guidelines" for using MAS like the MESSAGE project [24]. These approaches are beneficial and aid in the understanding of the problem, but it is conjecture that has not been demonstrated. The work in this thesis looks at the problem from a slightly different perspective.

In light of the fact that quantitative data does not exist to determine whether a MAS approach is superior to a more traditional object-oriented or functional approach, the goal remains the same—develop a high quality software product at the most reasonable cost. By looking at the problem from this perspective the focus is kept on the heart of software development, that is the engineering. Developing software by following a well-defined process is the best way to ensure that the ability to create quality software remains constant.

Narrowing down the question of *when to use a MAS* to *when to use an agent-oriented software engineering approach* is the first step in grasping this problem. Redefining the problem in this way, however, expands the task to include determining whether any type of software methodology is well suited to solving a problem.

When taking on a software requirement problem, a software engineer must decide how to go about solving that problem. Though the engineer may work in an organization that already has a well-defined process for developing software systems, the status quo may not always be the best approach—otherwise there would be no need for continued research in the software engineering field. To assist the decision-maker, this thesis develops a decision analysis tool, based on a mathematical framework, to quantitatively compare different methodology alternatives. In addition to developing a decision analysis tool that assists software engineers with determining an appropriate methodology amongst many different paradigms, the tool also provides a basis for comparing methodologies of the same paradigm.

5.2 Future Work

As agent-oriented methodologies continue to be developed, research will continue to look into the question of when it is the best time to use these methodologies. A number of areas of research into the question are needed. This section discusses a few of those areas as they relate directly to this research.

5.2.1 Basis for Decision

The first area of research suggested for future work considers the complexity of the decision that is being made. The basis for making the decision is broad. In this thesis, ten evaluation considerations are used to capture the differences between alternative methodologies. Though the decision to use the particular considerations came from literature sources and were backed up by the opinions of members of the multiagent system community, the farther the problem strays from those considerations, the less precise the decision analysis tool becomes. It is not unreasonable to think that a software requirement problem exists that would find many of the technical evaluation considerations irrelevant. In that case, the decision would be made wholly on the management considerations. From the data collected in Chapter IV, the alternative methodologies evaluated showed little variance in these evaluation considerations.

The flexibility of the tool allows for the integration of more evaluation considerations. Though it becomes more time consuming with each evaluation consideration to be measured, the decision maker is given the capability of selecting those considerations that are the most relevant to the problem at hand.

This flexibility allows for the potential creation of a library of evaluation considerations, which would enable the decision maker to customize the tool based on the particular problem.

5.2.2 Subjectivity

Another area where the evaluation considerations could be improved is to reduce the level of subjectivity in the rating. In this case, only one evaluation consideration, *Cost of Acquiring Methodology and Support Tools*, uses numbers from the physical world. As it is not likely that the quantitative data will exist in the near future, an alternative would be to conduct a research project that evaluates a large number of existing methodologies to build a database of scores for the particular evaluation considerations.

Research conducted by an individual or team would create a standardized set of data for rating the alternatives. Additionally, having this major data collection step performed *a priori*, the decision-maker would be able to include more considerations—from a potential evaluation consideration library—in the decision with a much smaller time requirement. As the set of considerations is increased, the basis for the consideration is improved.

5.2.3 Quantitative Data

Another area for future research is the generation of quantitative data that compares the life cycles of software projects developed via different methodologies. As shown in this thesis, small software projects can be developed by different methodologies and the resulting systems compared. The challenge, however, is to determine an appropriate method for collecting and comparing data on industrial-size systems. It is not the current practice of software development firms to simultaneously develop the same project following different paradigms.

This technique has been practiced in the past, however, in other engineering fields. Case in point is the Department of Defense's acquisition program in place to develop a Joint Tactical Strike Fighter. In this example, the U.S. Government is requiring fully operational prototypes before deciding which aircraft will be chosen.

5.3 Summary

This thesis defines a process of objectively evaluating alternative software engineering methodologies for a given software requirement. The challenge rests in the ability to transform subjective opinions into a mathematical framework. The first step in this process, as described in Chapter I, is developing a clear understanding of the aspects of the problem like decision-making processes, software engineering methodologies, and factors that are important to a general domain of software problems.

With a foundation of knowledge, the next step develops a process for achieving the hypothesis. Chapter II begins with the selection of an appropriate decision-making framework. The framework is further described throughout Chapter II and also includes the steps to develop the framework to fit the software engineering methodology selection problem.

The skeleton of the framework is refined throughout Chapter III. Real values are incorporated into the decision-making process via evaluation considerations such as the "Cost of Acquiring the Methodology and Support Tools." These objective numbers are combined with values from a subjective analysis of other evaluation considerations like "Methodology Maturity" and "Legacy System Integration." The fitness values of each evaluation consideration are combined to form a single multiobjective fitness value for the problem analysis.

Finally, Chapter IV presents a set of demonstrations over a number of representative problems. Validating the results of the decision analysis tool is a problem as difficult as creating the tool because once again, the availability of good, objective data is not available. For this purpose, one of the case studies was implemented following the two highest-ranking methodologies for comparison. The models, developed during the analysis and design phases of the methodologies, were presented to a panel of evaluators who were asked to determine the requirements of the system. Even this analysis demonstrated that comparing metrics is a subjective task and impractical on large-scale systems as the investment in dually developing systems is unlikely.

The decision analysis tool achieves the goal of determining the fittest methodology from a set of good candidates. Additionally, a basis for comparing methodologies within the same paradigm has been formed. The framework provides the decision maker flexibility by allowing for the inclusion and exclusion of specific evaluation considerations. Finally, even though the rating of criteria is highly subjective, the process ensures that a rationale and repeatable process for making the decision has been followed.

Appendix A. Background

A.1 Overview

Multiagent systems are a relatively new paradigm for solving complex, distributed-processing software problems. By providing a new way of looking at the problem, agents offer a unique set of advantages to help solve the problem. Agents also present a set of problems, which may require the software engineer to make use of other techniques to build software solutions. The purpose of this appendix is to provide a survey of background information on several topics crucial to the development of this thesis. Section A.2 begins with an investigation of the characteristics of agent systems. Section A.3 compares and contrasts three major software engineering paradigms including the agent-oriented approach. Next, Section A.4 surveys of the pitfalls and problems with multiagent systems and the agent-oriented software engineering approach in general. Section A.5 discusses current research that is related to this thesis. Finally, Section A.6 explores techniques for making decisions.

A.2 Multiagent System Properties

Before multiagent systems can be fully explored, a common vernacular must be shared. Though there are differing opinions of how certain terms should be defined, most definitions have some common characteristics. This section begins with definitions of terms that have been used throughout the thesis. Following the definitions, different properties and characteristics of multiagent systems will be explored.

A.2.1 Definitions

Below are a number of terms that are basic to the study of agent systems and artificial intelligence. The source of these definitions is [40].

Agent: A computer system that is situated in an environment and is capable of autonomous action in the environment in order to meet its goals.

Autonomy: The ability to act without outside intervention. That is an agent has control over its internal state and behavior.

Flexibility: With regard to agent intelligence, flexibility is comprised of reactivity, pro-activeness and social ability.

Intelligence: The ability to achieve goals in a flexible and autonomous manner.

Pro-active: The ability to exhibit goal-directed behavior by initiating actions that work toward satisfying a goal.

Reactive: The ability to perceive the environment and respond to changes in the environment in order to achieve a goal.

Social Ability: The ability to interact with other agents in order to achieve a goal.

A.2.2 Environments

The environment in which the agents exist is an important factor in agent properties. By considering two different environments—closed and open—these properties can better be evaluated. In a *closed* system environment, the system designers know exactly what type of agents will exist. For the most part the agents will be *cooperative*, that is, the agents will work together toward a common goal. In an open system environment, agents are likely to meet both cooperative agents and competitive agents. *Competitive* agents are self-interested; they have their own set of goals, which may be in conflict with the overall goal of the system. Self-interested agents will assist with the groups problems, usually at some cost, which is negotiated prior to performing any computation. For an agent to work within an open system, the agent also must know how to handle communications with agents that were unknown to the system designer when the agent was developed [9].

A.2.3 Problem Domains

Problem domains that multiagent system approaches are suited to solve generally have an inherent form of distribution; that is, knowledge, capability, information, and expertise are all resources that can be distributed throughout the system [3]. Three examples of problem domains include distributed situation assessment, distributed resource scheduling and planning, and distributed expert systems [23].

Distributive situation assessment, or distributed capability, deals with agents combining partial information to solve complex problems. The information comes from separate areas so that a single agent is not able to gather everything by itself. Multiple information gatherers must combine their information to see the "big picture." A distributed sensor network establishment is an example of distributed capability. In a distributed sensor network establishment, a large area is monitored for, say, vehicle movement. The area is too large for a single sensor, so the overall task of monitoring the area is distributed across several agents. The information each sensor receives must be synthesized with the information from the other sensors to completely track the vehicle movement [3].

The distributed resources scheduling and planning example deals with maintaining coordination and conflict resolution over a set of resources. Generally, in this type of system, an agent represents each resource. The overall system goal is to maximize system output. Distributed delivery agents are an example in this case. The planning or the execution of a plan can be distributed across many agents [3]. Additionally, the planning and the execution of the plan may be occurring simultaneously, and as the environment changes the agents may be required to unexpectedly replan.

Distributed expert systems are the third example. This example also deals with how multiple agents collectively gather the information, however, in this case, each agent has some expertise to offer the problem solution, and coordination of partial solutions is required [23]. Additionally, expert systems may be too large or too expensive to replicate throughout the system [3].

Large-scale applications that have been developed such as ADEPT—a business process management system—and ARCHON—an electricity transportation management system—have the following characteristics [14]:

- Each agent makes use of a significant computational resource, such a UNIX process
- Agents are cooperative, generally maximizing a system goal over individual goals
- Agents are heterogeneous in implementation
- The system as a whole contains a small number of agents

A.2.4 Benefits

By building systems that involve multiple agents, the system should realize benefits such as speed up due to concurrent processing, less communications bandwidth requirements as information gathering is performed at the source and not remotely, and increased reliability since there is not a single point of failure. Other benefits include increase responsiveness due to processing and sensing being performed at the source, and easier system development as each agent is modular [23].

A.2.5 Principles

When designing multiagent systems, Lesser points out a number of useful principles. First, the ability to maximize all of the system goals is unrealistic [23]. The complexity involved in finding optimal solutions for every agent, particularly when agents may have conflicting goals, is extreme. Instead, the system should attempt to find a realistic solution within a reasonable amount of time using a reasonable amount of resources. This is called *satisficing* a goal, as it is an acceptable answer generated within a reasonable amount time. The second principle is the need for flexibility in problem solving. This flexibility involves access to resources or other agents that are dynamically available. The ability to find alternate problem solving capabilities allows the system to be a robust problem solver. The third principle is the ability to exploit the organizational structure of the agent system. Computation needs to be distributed equally across the organization, though the type of computation may be different. At the bottom of the organization, the computation may involve low level functional processing, where at the higher level, over-sight and coordination jobs may be executing [23].

A.3 Software Engineering Paradigms

Over the last four decades, software has become an important piece of information systems. During this time, software engineering paradigms have been designed as methodologies for creating good software in a reasonable amount of time and at a reasonable cost. These paradigms are generally layered approaches to developing solutions from a customer's non-technical description of what the system must be capable of doing. IEEE defines software engineering as

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1). [10]

With this definition of software engineering in mind, three modern paradigms are examined and discussed: Object-Oriented, Component-ware, and Agent-Oriented. Each of the engineering paradigms offers systematic approaches to, yet a different perspective of, the basic approach to building software. All three approaches can make use of object-oriented programming languages, such as C++ and Java. In some cases the differences the paradigms exhibit are quite substantial, and in others they are minor. Though the paradigms are different, they are designed to assist humans in the complex task of software development.

The complex nature of software exhibits a number of consistent characteristics. Because these characteristics are consistent over most software problems, software engineering provides techniques and structures for handling these problems [31]. The first characteristic that complex problems exhibit is that the problem generally takes the form of a hierarchy. The problem can be broken down into sub-problems that when solved individually can be combined to solve the entire problem. A second characteristic of complex problems is that the selection of the basic building blocks of the systems is arbitrary and based on the designer's goals. A third characteristic is that the hierarchical form of the problem allows for quicker development because intermediary forms of the solution are more stable. Finally, the fourth characteristic is the ability to distinguish between the communication among subsystems and those within the subsystem.

Again, each of the three paradigms addresses these four characteristics. The important thing to remember, however, is that all three of the paradigms are legitimate ways of solving software problems.

A.3.1 Object-Oriented Paradigm

Before discussing the paradigm, it is imperative to understand what an object is. The term *object* is one of the most loaded words in Computer Science. It began emerging simultaneously in different circles in the 1970's and still has many connotations [2]. Listed below are some of the derivative terms that have become part of the computer science vernacular.

- *Objects*: Objects are atomic entities that encapsulate identity, state and behavior [28]. Software objects can be thought of simplified abstractions that represent any real-world entity. The properties of the world the object represent are generally the aspects of the entity that are necessary to the user [25].
- *Active Objects*: Active objects have emerged in the field of object-oriented concurrent programming. The concept of the active object is to merge the attributes and methods of an object with an autonomous activity. Behaviorally, active objects act the same as objects; they react only to method call and remain procedural [7].
- *Distributed Objects*: Distributed objects are described as "independent pieces of code that can be accessed by remote clients via method invocation"[30]. In one sense they are components (see section A.3.3 for more information on components). The language and compiler used to build the distributed object are transparent to the user. The only requirement to use the object is to know how to invoke the remote method. Additionally some concept of a service broker must be established for clients to find the distributed object.

Of the three paradigms presented, object-oriented is the oldest. Booch, an early advocate of object-orientation, describes the complexity of industrial strength software systems as being beyond the ability of a single person to fully comprehend [2]. He notes that experiments by psychologists indicate the most an individual can simultaneously comprehend is about seven pieces of information. With these limits in mind, Booch suggests the concepts of decomposition, abstraction, and hierarchy as ways of dealing with the complexity.

Decomposition is a method of divide and conquer. It allows the designer to take a broad problem and transform it into a set of sub-problems. Each of these sub-problems can be further refined independently. Thus, decomposition allows the user to solve sub-problems that are manageable. Object-oriented decomposition breaks the problem down into the objects. Techniques described by Pressman and Muller suggest deriving objects from the nouns in the problem specification [25, 31].

Abstraction allows the user to remove many of the details from the objects, capturing only the most significant. This also reduces the complexity and makes the problem more manageable for the

designer. Common analysis and design techniques are iterative and more detail is added as the problem goes through successive iterations.

Hierarchy also plays an important role in making the problem less complex. The recognition of hierarchical structure between objects increases the semantic content of the information. Capturing the hierarchical structure between objects is not always easy, however, as each object may exhibit complex behavior.

Object-oriented software engineering was designed to exploit the features of object-oriented programming techniques. Objects, represented in languages such as C++ and Java, encapsulate all of the details the object exhibits in the "real" world, or at least the details relevant to the problem. These details include attributes—for example a person object might have the attributes name, age, and sex—as well as actions the object can perform, called methods. Objects can represent physical objects in the world, such as people, cars, and rooms, but they can also represent abstract objects like a schedule.

Object-oriented software engineering assists the software engineer with the task of addressing the concepts of modern software engineering: information hiding, data abstraction, encapsulation, and concurrency. These concepts are not easily handled with the structured analysis techniques developed prior to object-orientation. For example, the ability to encapsulate actions and attributes that belong to a particular object provides a more semantically correct model than just a function. Likewise, it allows for dealing with entities that are larger than sub-routines. This, in turn, allows for better management of the complexity of the system. Object-orientation also promotes the concept of software reusability. As the functions and attributes of the entity are encapsulated within the object, the object is easily migrated to other systems for reuse. Incorporating an existing function is much more difficult as the function was designed to fit within a specific task. An object-oriented application is a closer representation of the real world than a functional application. The ability to better understand the mapping of the real world to the software abstraction greatly improves the users confidence in the system. Additionally, the ability to understand the code is greatly enhanced by objects-orientation, which in turn leads to better maintenance and improved modifiability. Since objects provide a certain level of data abstraction and information

hiding, the underlying functionality of the object can remain the same, but the implementation can easily be changed to provide improved performance or additional functionality.

A.3.2 Object-Oriented Methodologies

Over the past decade, object-oriented methodologies developed from two schools of thought. The first, *revolutionary*, represents a radical change over the traditional methodologies, effectively rendering the traditional methodologies obsolete. The second school of thought is *synthesis*. The synthesis point of view sees object-orientation as an evolution of the conventional methodologies [4]. A number of researchers developed analysis and design techniques for creating object-oriented software from both of these schools of thought.

In all, approximately fifty object-oriented methods emerged [25]. As these methods became further developed, the differenced between the methods grew smaller. Figure 52 shows some of the key elements that came from this work. In 1994, Rumbaugh and Booch began work on unifying their work into a single method. The Unified Method would come to bring these different models together and standardize the notation.

Origin	Element
Booch	Categories and subsystems
Embley	Singleton classes and composite objects
Fusion	Operation descriptions, message numbering
Gamma, et al.	Frameworks, patterns and notes
Harel	Statecharts
Jacobson	Use Cases
Meyer	Pre- and post-conditions
Odell	Dynamic classification, emphasis on events
OMT	Associations
Shlaer-Mellor	Objects' lifecycles
Wirfs-Brock	Responsibilities and collaborations

Figure 52: Origin of Object-Oriented Methodologies [25]

The convergence of the notations was a difficult task, as many people had input to provide, After the release of the Unified Method v0.8, more than a thousand detailed comments were returned to the authors. The Unified Method underwent several substantial evolutions, which can be seen in Figure 53. The result of this effort is now known as the Unified Modeling Language (UML).

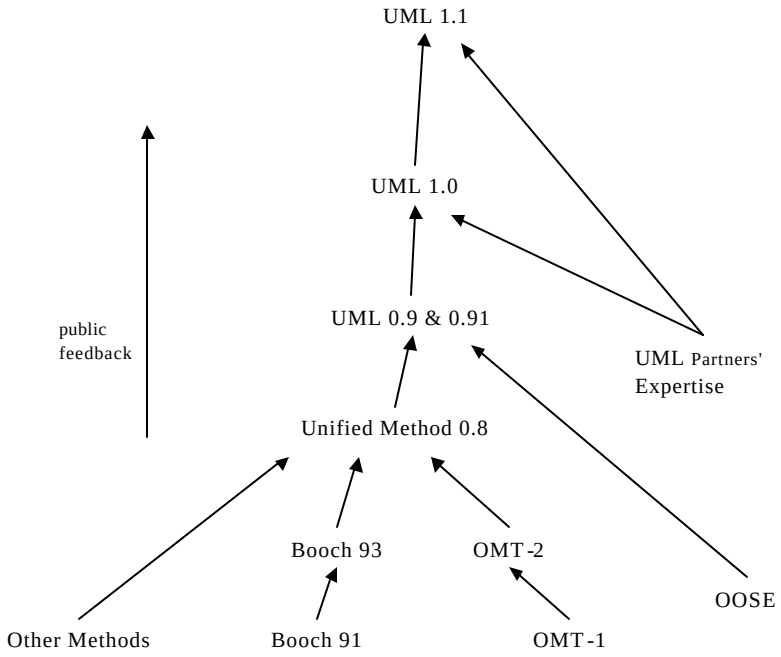


Figure 53: Development of UML [25]

A.3.3 Component-ware Paradigm

The component-ware paradigm is an attempt to maximize software reuse. Components are defined by Szyperski to be binary units of independent software [37]. The term binary refers to the component being executable. Components usually perform a single function and are treated as "black boxes" [34]. A common analogy compares software components to integrated circuits. One major difference between software and hardware is that software components have no limit to the number of instances that can simultaneously exist.

The original creation of components can come from any software engineering technique, such as object-oriented or structured analysis. Internally, the method of storing data is irrelevant to the user. Any

information that is maintained by the component is only accessible by prescribed methods provided by the component. The versatility of components is, therefore, great. The driving purpose behind developing and using components is software reuse.

In order to reuse the components, Szyperski introduces "contracts" as a specification for the component [37]. The contract describes the required input for the component and a guarantee on the output. It is important to only rely on the contract for expected behavior, as future versions of the component may not implement the component in the same way. The reuse of components is realized in different ways. Components can provide a purely functional service, like a hardware component, where some piece of information is passed to the component, which then evaluates, manipulates, or transforms the information. The fact that the details of how the function actually works are hidden allows for the component to be switched with other like-function components, which provide the same service, only implemented differently. For example, a component that provides a list-sorting algorithm may be designed to implement a selection sort algorithm. In a future release of the component, the implementation may be a quick sort algorithm. Since the requirement for input to either component is the same and the result will be a sorted list, the actual implementation is a moot point to the system designer. That is not to say that there are not other factors that go into component selection, as in the previous example, the quick sort algorithm will likely have better performance than the selection sort algorithm, but the cost of the quick sort algorithm may be prohibitive.

Another way components can be used is as objects that can be used for inheritance. The attributes and the methods that a subclass inherits from the super-class component must be defined in the contract, but the details do not. Examples of object components are Sun's Java Beans [37]. Java Beans have been developed for creating graphical interfaces. The details of how the Beans are displayed are completely hidden from the user, yet the Bean can be extended to provide a more specific look.

Systems are built by combining components. This component composition requires connecting interfaces. Components developed in an object-oriented method may provide for inheritance, thus allowing the system to make use of the methods within the component. Functionally developed components make

use of parameterized function calls. In some instances, a scripting language may be needed to act as the *component glue* [37].

With regard to the three methods of dealing with complexity—abstraction, decomposition, and hierarchy—component-ware systems generally make use of an analysis and design methodology like object-oriented or structural decomposition. As such, the component satisfies the level of abstraction desired by the designer. Additionally, when a component is selected in the design phase, the problem has been appropriately decomposed to the level that the component satisfies the goals of the subsystem. Likewise, the component fits within the hierarchy of the system without having to provide further detail.

A.3.4 Component-ware Methodologies

It is not likely that one would be able to compose a system strictly from a library of components, even if a large library was available. As components are developed as "a physical, replaceable part of a system" [19], the software developer must make use of other techniques to generate the non-component portion of the software. The technique must be able to incorporate the component through the components specified interfaces.

In version 1.3, the UML semantics has been expanded to include the modeling of components. By expanding the UML semantics, a software engineer can follow the object-oriented methodology while gaining the benefit of software reuse.

A.3.5 Agent-Oriented Paradigm

Agent-oriented software engineering is an emerging paradigm based on the object-oriented paradigm. Like the object-oriented paradigm, the agent-oriented paradigm is designed to reduce the complexity through decomposition, abstraction and hierarchy. Jennings suggests using the term organization, instead of hierarchy, however, as it is a more neutral term without connotations of control. Though organizations can correspond to hierarchies, they can also correspond to peer groups, or anything that falls within that spectrum [15].

As expected, the primitive building block in the agent-oriented paradigm is an agent. Agents are more expressive than objects. Every agent is autonomous, having control over its own internal state and behavior and having well-defined boundaries and interfaces. Other characteristics of agents include the ability to exhibit flexible problem solving capability, fulfill a specific role, and exist in an environment whereby they can detect changes through sensors and make changes through effectors.

By adopting an agent view of the world, it is apparent that most problems require multiple agents to represent some decentralized nature of the problem, multiple control centers, or multiple goals [14]. One difference between agents and objects is their interaction techniques. Objects interact with one another through method invocation, whereas agents communicate through a high-level agent communication language (ACL). ACLs allow the interactions to be conducted at the knowledge level. By using an ACL, agents do not need to know anything about the structure of other agents. The only requirement is that the agents know how to pass and accept messages from other agents.

In Section A.3.1, active objects and distributed objects were introduced. One characteristic that differentiates these types of objects from agents is their communication technique. Both types of objects are monolithic and rely on direct method invocation. Additionally, the agents can exhibit many behaviors autonomously, where as an active object typically has a single behavior. Research has looked at extending the active object notation in UML for agent analysis and design [7].

A.3.6 Agent-Oriented Methodologies

Like object-orientation, agent-orientation can be viewed as a revolutionary or evolutionary step in software engineering. In Iglesias et al.'s survey of methodologies, most agent-oriented methodologies are extensions of the object-oriented and knowledge engineering methodologies [11]. The reason most methodologies are based on existing methodologies is because developers already know some of the vocabulary and techniques. The parts of the methodology that do not apply are removed, and aspects that are not addressed are added to the new methodology.

Wooldridge, Jennings, and Kinney have developed an analysis and design methodology called Gaia [42]. The MaSE methodology is also a complete analysis and design technique for developing agent

systems. These methodologies are intended to capture an agent's flexible, autonomous behavior, the interactions between agents, and the complexity of the agent system's organizational structure. These intentions are fulfilled by the methodologies' modeling of system goal, the roles and behaviors that the agents exhibit, and the communication protocols between the agents.

A.3.7 Paradigm Comparisons

Research at AFIT by Robinson has already begun looking at agent assembly from a component point of view [32]. This approach has much merit, as it is common for system designers to add features through new system releases. To be able to add and remove agents from a system without having to redesign the entire system allows for low cost reuse. Additionally, as long as the agent "contracts" clearly define the agents communication language and the services it provides, system composition becomes fairly trivial, not requiring any type of component glue [37].

From Sections A.3.1 through A.3.4, it is clear that there are many similarities between the agent-oriented paradigm and the other paradigms. There are some important differences, though, that are not quite as obvious. The remainder of this section will highlight those most important differences.

Booch describes objects as "a set of autonomous agents that collaborate to perform some higher level behavior" [2]. This characterization of objects as agents is not far from the truth, but there are some differences between agents and objects. Objects and agents do both encapsulate the who, what and how. Agents, however, encapsulate more. The knowledge they encapsulate includes the when—their own thread of control—and the where—the location of the execution. Additionally, they contain why they do something, that is, a software agent captures the goals that it has been design to accomplish. With an interface to the environment and other agents, contracts and services are negotiated at a meta-level [28].

Intelligence and autonomy are important characteristics that separate agents from objects and components. Objects and components are typically passive and obedient. They do not perform functions unless specifically asked and have no ability to refuse a request. Even active agents are limited in their autonomy. For example, active agents are monolithic and only encapsulate a single behavior.

Additionally, active objects do not reason about its behavior or interactions with other objects [37]. Agents, on the other hand, can exhibit many behaviors as well as choose to perform a task. The decision is generally made by a utility function that may take into account such things as the cost of the operation, the current workload, and the expected pay back. One concept not addressed by the object-oriented paradigm was the agent's mental state, or internal view of the environment. Objects maintain attributes that are characteristic of the object itself and provide the operators to effect those attributes. Extensions to the object-oriented paradigm need to address issues of inference, planning, etc. [28].

In large complex systems, direct method invocation requires the requesting entity to know much more about the entire system. Traditional distributed software can use techniques such as remote method invocation or message passing. In a system developed in Java, the system can implement the remote method invocation via RMI or CORBA, and it can implement message passing through sockets. In both cases, the object requires some *a priori* knowledge. For example to create a socket connection, the local class needs to know the host name, or address, and the port number of the distant class. Likewise, using RMI or CORBA, class stubs must be generated for both client and server and placed on the respective systems, as well as locally running a registry with knowledge of the location of the objects. In an agent system in which agents can broker and negotiate for services, entities can join dynamically [15]. That is, agents are free to initiate action independent of any other entity [28], though some *a priori* knowledge is needed with regard to a service server. The agent is not required to making use of a single service server, however. The agent can maintain a list of service servers and dynamically select the server with which to negotiate.

Components are not as powerful as agents computationally. Components are passive, requiring invocation in order to provide service. Additionally, a component's contract specifies the preconditions the incoming data must meet in order to guarantee the correctness of the output. For components to be useful, they must be general problem solvers. Components that are loaded with too much functionality become expensive, especially if some of that functionality is not going to be used [37]. Agents, however, can be

designed with the ability to find or request services within the system that they cannot provide, thus allowing a dynamic problem solver [15].

A.4 Agent-Oriented Problems and Pitfalls

The focus of this section is to highlight the downside to taking an agent-oriented approach to software engineering. To date, the main problem identified is the great difficulty software engineers have in validating their systems due to agent and system unpredictability. In addition to these problems, a number of pitfalls, as presented by Wooldridge and Jennings are summarized. The significance of identifying these problems and pitfalls is to the software engineers' advantage for the purpose of selecting a software engineering paradigm.

A.4.1 Agent-Oriented Problems

Current research has highlighted two major downsides to the agent approach for software development. Those downsides are intrinsic to agent interaction. Since agent systems exist, software engineers have been able to work around these problems, however, the fixes are made on a case-by-case basis and there is not currently a general solution. These problems can be seen at two level of the system: the individual agent and the agent system.

At the first level, the agent level, agents exhibit a natural unpredictability. Being able to perceive changes in the environment, agents must balance proactive and reactive behavior. By being too proactive, the agent may undertake tasks that are, or may become, irrelevant. On the other side, an agent that is too reactive may spend too much time fulfilling requests and may not fulfill its own goals. By reaching a compromise, agents demonstrate a new level of sophistication and flexibility. The only problem with this is that they also are quite unpredictable. Because of the vast number of combinations of events and changes to the environment that can exist in an environment, it would be impractical to try to test every combination of events and actions. Since the decision to work toward a particular objective is decided at run-time based on the state of the environment, the ability to predict the agent interactions is extremely difficult. Furthermore, if agents are given the ability to negotiate for services and tasks, the service that an agent requests and the service that is actually performed on its behalf cannot be predicted *a priori* [14].

The second level, the agent system, is where another source of unpredictability can be found. At this level, the unpredictability is associated with emergent behavior. As a benefit to multiagent systems, emergent behavior is a synergetic effect that makes the system more powerful than the individual agents combined. Another way of interpreting this effect that demonstrates the problem with emergent behavior is well stated by Jennings, "results in behavioral phenomena that cannot be deconstructed solely in terms of the behavior of the individual components" [14]. Just as with the individual agent, emergent behavior makes the ability to reproduce or predict the system interactions very difficult.

A.4.2 Agent-Oriented Pitfalls

In attempt to understand how to better engineer software projects using an agent-oriented approach, it is necessary to consider the possible problems that can arise. Furthermore, there are other factors that determine whether or not the agent-oriented paradigm will succeed. Presented here is a summary of the main pitfalls as seen by Wooldridge and Jennings [41]. They identify seven categories in which twenty-four key pitfalls have been classified.

?? *Political Pitfalls* – The two pitfalls that fall under this heading are very similar to problems that have plagued the field of Artificial Intelligence as a whole. The first is overselling agents as a solution. Agent-oriented software is not promising to provide solutions to problems that other software engineering approaches cannot. There is currently no data to suggest that an agent-based solution could not be developed in a non-agent form. The second pitfall in this category is being dogmatic about agents. This technology is still immature and the use of it should be based on a clear advantage. If a system can be designed with a non-agent approach and perform just as well as an agent approach, then the non-agent approach should be selected. The argument here is that until the agent more engineers better understand approach, an agent system will be harder to maintain.

?? *Management Pitfalls* – This category has four key pitfalls. The first two pitfalls involve ignorance of agents. Management does not know why they need agents or what the agent is good for, but they understand that it is a state-of-the-art technology that is currently hot in

today's market. The problem with using agents when they are not really understood usually involves inappropriate use of the agents. When the system does not perform correctly the use of agents is blamed and the reputation of the technology is tarnished. The other two pitfalls in this category involve the system design. With regard to re-use and building "one-off" solutions, general solutions are difficult and costly to design and implement and usually do not live up to expectations.

?? *Conceptual Pitfalls* – There are four conceptual pitfalls that involve fully understanding the software engineering process. The agent-oriented paradigm does not offer a "silver bullet" solution to software engineering's greatest challenges. Misunderstanding the concepts that go into agents is another major problem. The idea of an agent is intuitive, but that should not mean developers should not study the concepts in depth. Software engineering is a complex process, and many well-founded techniques for doing so have been developed. Forgetting these techniques can be disastrous for software development.

?? *Analysis and Design Pitfalls* – Two pitfalls fall under the analysis and design category. One involves the using current technology to the agent's advantage. The "agent-specific" aspects of a system are small compared to the whole system, so it is vital to use the state-of-the-art technology in the non-agent areas. Failing to exploit concurrency is the second pitfall. Multi-agent systems are well suited for distributed problems that can be solved concurrently. Concurrency allows for multiple objectives to be simultaneously achieved.

?? *Agent Level Pitfalls* – The pitfalls that fall under this category involve internal agent design. With regard to using artificial intelligence techniques, agents can have too much or not enough. Putting in too much intelligence can hinder the performance of the agent as well as overburden the processor the agent is running on. Agents that have no intelligence are bad because they make the term "agent" lose it's meaning. Additionally, "agents" raise certain expectations that could leave users dissatisfied if the non-intelligent system does not live up to the expectations.

?? *Society Level Pitfalls* – Under this category, six pitfalls have been identified. These pitfalls involve incorrectly identifying agents within the system. Using too many agents can drive the communications overhead up, where as using too few agents can improperly divide the workload. Another major pitfall is confusing simulated and real parallelism. As a multi-agent system prototype may provide a good insight to distributed problems, the system running on a real network may show much different results than the system running on a scaled-down isolated test network.

?? *Implementation Pitfalls* – The two pitfalls that fall under this category involve misconceptions of using a new technology. Building every system from scratch can drive up the cost of software development. Though some standards are not firmly established, the groundwork for the standard has been laid and developers should attempt to work within that framework.

A.5 Related Work

As agent-oriented software engineering develops into a viable technique for solving software problems, the question of what type of problem is suited to the technique arises. This question is similar to the problem that this research is addressing, but the current work that has been done looks just at the agent-oriented side of the problem. Two major undertakings with regard to answering the question of for what type of problem is the agent-oriented approach appropriate are described below.

A.5.1 Jennings and Wooldridge's Work on Agent-Oriented Software Engineering

Jennings and Wooldridge have presented numerous papers on agent-oriented software engineering. Their work presents a qualitative analysis of the engineering approach to provide "intellectual justification" [15]. Their work is an attempt to answer why agent-oriented software engineering is the "new revolution in software engineering" [15]. The argument that they make is based on three parts [15]:

1. Agent-oriented decompositions are effective ways of partitioning the problem space of a complex system.

2. The key abstractions of the agent-oriented mindset are natural means to abstracting a complex system.

3. The agent-oriented philosophy for identifying and managing organizational relationships is appropriate for dealing with the interactions and dependencies in a complex system.

By arguing these issues, Jennings and Wooldridge assert that the agent-oriented techniques represent an advance over the current state of the art. Based on these three areas, they compare the agent-oriented technique to object-oriented and component-based software engineering. They leave as "outstanding issues" the following [15]:

- Understanding of the situations in which agent solutions are appropriate
- Principled development techniques for agent systems

A.5.2 MESSAGE: Methodology for Engineering Systems of Software AGEnts

MESSAGE is a project that is being developed by the European Institute for Research and Strategic Studies in Telecommunications (EURESCOM). Initiated in 1999 as a two-year project, seven major European telecommunication industries formed a consortium to achieve three objectives [24]:

- ?? Define a suitable methodology for agent-based application development in the telecommunications domain
- ?? Identify and recommend tools to support the methodology
- ?? Define guidance for the identification of application areas where an agent-based approach is better suited than other approaches

The aim of the project is to extend existing approaches, such as UML, to allow them to support Agent-Oriented Software Engineering (AOSE). Furthermore, MESSAGE is attempting to combine the best features of other AOSE approaches that are currently being defined.

The particular interest with the MESSAGE project with regard to this research is its focus on its third objective. Currently, they are promoting five guidelines that are intended to help the developer in deciding whether or not an agent-oriented approach is appropriate. Those guidelines are presented below [24]:

1. An agent-oriented approach is beneficial in situations where complex/diverse types of communication are required.

2. An agent-oriented approach is beneficial when the system must perform will in situations where it is not practical/possible to specify its behavior on a case-by-case basis.

3. An agent-oriented approach is beneficial in situations involving negotiation, co-operation and competition among different entities.

4. An agent-oriented approach is beneficial when the system must act autonomously.

5. An agent-oriented approach is beneficial when the system is expected to be expanded or modified or when the purpose of the system is expected to change.

Further development of these concepts is the topic of the projects second deliverable, which has not been published at this time. Based on these five guidelines alone, there is still no clear answer as to whether or not an agent approach is appropriate. Does a system have to meet one guideline, two, or perhaps all five to be well suited to the approach? Likewise, the guidelines do not address other factors that may argue against an agent-oriented approach even if the system meets all five guidelines; hence, the necessity of this research.

A.5.3 Methodology Selection for Developing Real-Time Systems

In 1988, the Software Engineering Institute presented a set of guidelines for assessing software development methods for real-time systems [38]. The guidelines are a five-step process for evaluating different methodologies. The five steps of the guidelines from [38] are given below:

1. *Needs Analysis – Determine the important characteristics of the system to be developed and how individual methods help developers deal with those characteristics.*

2. *Constraint Identification – Identify the constraints imposed on the permitted solutions and determine how individual methods help developers deal with those constraint .*

3. *User Requirements – Determine the general usage characteristics of the individual methods.*

4. *Management Issues – Determine the support provided by the method to those who must manage the development process as well as the costs and benefits of adopting and using the method.*

5. *Introduction Plan – Develop an understanding of the issues that the method does not address and a plan to augment the method in those areas where it is deficient.*

With these steps, the authors present questions for the software engineer to answer about the methodology. Some of the questions are rhetorical, meant to remind the engineer of the issue. Other questions require an in-depth knowledge of the methodology and its representations. The purpose of the questions is to make the assessor form an opinion regarding the methodology, however, not over-simplify the problem of selecting a methodology. The questions provide a framework to present a systematic evaluation process.

A.6 Decision-Making Frameworks

A goal of this research is to specify a decision-making process for determining an appropriate software engineering technique. Clearly, the decision to select one methodology over another is a difficult

tasks where many trade-offs must be made. As such, a multidimensional technique for decision-making is required. This section explores two different techniques for making multidimensional decisions.

A.6.1 Design Space and Rules

In the field of software architectures, the concept of design space and rules has emerged as a methodology for describing and classifying architectural alternatives and indicating good and bad combinations of those alternatives [21]. The intention of the design space and rules methodology is to assist new engineers in making the correct choice in design, as would an experienced software engineer. This technique is the foundation for a decision-making process for software engineers when trying to determine which design options will provide a good solution to a given requirements statement.

The design space is a multidimensional space for classifying systems. Each dimension of the design space represents different structural and functional characteristic of the system. Structural dimensions represent characteristics that pertain to the techniques that can be employed, where as the functional dimension captures the impact of the decisions. An example of two dimensions is response time (functional) and interprocess synchronization (structural). A point in the design space then determines a design. This small design space is illustrated below in Figure 54 [21].

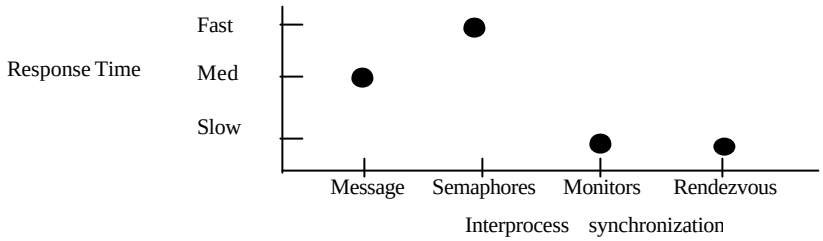


Figure 54: A Simple Design Space

Not all dimensions are independent of one another and it is important to find these correlations. As correlations between alternatives are discovered, rules can be generated to assist in the decision process. For example, a correlation that can be drawn from the simple design space in Figure 54 is the relationship between interprocess synchronization and response time. If the specification required a highly responsive

system, then messaging or semaphores would be better design choices that monitors or rendezvous. Another functional dimension that the "Interprocess Synchronization" may be plotted against is complexity. The space may show that messaging and semaphores are "very complex." The software engineer is now required to balance the complexity of the system with the responsiveness. By testing existing systems against the design space and rules, the degree of agreement—that is how the actual implementation compares to the design predictions—can be measured.

Selecting the dimensions that reflect the requirements and describe the structure of the system creates a design space. Using the correlations of the design space, the designer is able to directly see which choices meet the requirements of the specification. Rules can be developed based on the design space and the designer's experience. These rules could be automated to accept a set of requirements and propose a good, but perhaps not optimal, design.

By defining a set of dimensions that can be used to establish a system design, existing systems could be tested against the design prediction to determine a degree of agreement. Systems that score high degrees of agreement will validate the design prediction and the rules that led to that prediction. Additionally, systems can be modified to test different combinations of the dimensions. These modified systems' degree of agreement can also be determined to further test the design rules.

At AFIT, design space and rules were used in the integration of software engineering tools [26]. In this research, Noe demonstrated the possibility of integrating different software engineering tools based on the tools extendibility and communication paths. Noe's design rules were used as guidelines for selecting options in the structural dimensions based on the requirements given by the set of functional dimensions. With these guidelines in place, formal transformations were developed. New tools could be integrated with each other by determining the tools structural characteristics and selecting the appropriate transformations.

A.6.2 Multiobjective Decision Analysis

The purpose of using a strategic decision making technique is to apply a quantitative approach to a problem that can be greatly qualitative. Using numbers to quantify choices clarifies different elements of the decision and makes the user of the technique be explicit about reasoning. In turn, the basis for a decision can be reviewed and analyzed during and after the process. Additionally, communication between different interested parties is improved, as there is a concrete baseline for discussion [18]. Strategic approaches to decision making generally follow the five phases shown in Figure 55.

1. Specify objectives and scales for measuring achievement with respect to these objectives
2. Develop alternatives that potentially might achieve the objectives
3. Determine how well each alternative achieves each objective
4. Consider tradeoffs among the objectives
5. Select the alternative that, on balance, best achieves the objectives, taking into account uncertainties

Figure 55: Five Phases to Strategic Decision Making [18]

This section begins with definitions relevant to decision making. Following the definitions, the five-phase strategic decision making process is dissected and the critical points in each are discussed.

A.6.2.1 Taxonomy

This section defines a set of terms that are used throughout the thesis for the purpose of discussing the multiobjective decision analysis technique. The definitions of the terms are derived from [18].

- *Alternatives:* Alternatives are different realizations of a decision. The purpose of a decision-making technique is to assist a person in selecting the best alternative to a choice.

- *Evaluation Consideration:* An evaluation consideration is a matter that needs to be taken into account when an alternative is being evaluated. For example, evaluation considerations for a person that is

looking for a new job may include salary, location and advancement opportunities. Evaluation considerations are also known as *evaluation concerns* or *areas of concern*.

- *Evaluation Measure:* An evaluation measure is a scale for the degree of attainment of an objective. For example, "annual salary in dollars" may be an evaluation measure for a job seeker looking for a new job with a higher salary. Other terms used for evaluation measure include *measure of effectiveness, attribute, performance measure,* or *metric*.

- *Goal:* With respect to an evaluation consideration, a goal is the threshold of achievement that is either obtained or not by an alternative. For the job seeker example, a goal may be a salary of $50,000. A goal is also referred to as a *target* [16].

- *Layer or Tier:* The evaluation considerations at the same distance from the top of a value hierarchy make up a layer or tier. The evaluation considerations in layers higher in the hierarchy are composed of the evaluation considerations lower in the hierarchy.

- *Level or Score:* The level or score represents the specific numerical rating for a particular alternative with respect to a specified evaluation measure.

- *Multiobjective Value Function:* A multiobjective value function is the combination of all of the evaluation measures within a value structure to derive a single value.

- *Objective:* An objective is the preferred direction of movement with respect to an evaluation consideration. Typically, the preference will display monotonic behavior. An example of an objective of the job seeker would be to have a *higher* salary.

- *Single Dimensional Value Function:* A single dimensional value function, or single attribute value function, is a function that returns the level of an evaluation measure based on the alternative.

- *Value Hierarchy or Value Tree:* A value structure with a hierarchical or tree-like structure is a value hierarchy or value tree. The structure can be presented textually or graphically.

- *Value Structure:* The value structure is the entire set of evaluation considerations, objectives, and evaluation measures for a particular problem.

-*Weights:* A weight is used in a multiobjective value function to place more or less emphasis on a particular evaluation measure. The value of the weight is multiplied by the level of the evaluation measure for a particular alternative. The sum of the weights in a multiobjective value function must equal one.

Note: In the literature, the use of the terms "goal" and "objective" are sometimes interchanged, but this research will follow the definitions presented here.

A.6.2.2 Value Hierarchies

The first step in the five-step strategic decision-making process is to capture the objectives of a decision and specify evaluation measures with respect to the objective. A value hierarchy allows a decision maker to structure evaluation considerations, objectives, and evaluation measures in a structured and understandable format.

Value hierarchies are generally depicted as tree diagrams with the root of the tree either at the top of the diagram with the children nodes below, or at the left of the diagram with the children nodes to its right. Value hierarchies can also be represented textually in an outline format. The value diagrams that will be used in this research will be graphical with the node on the right. An example of a value hierarchy for our job seeker is shown in Figure 56.

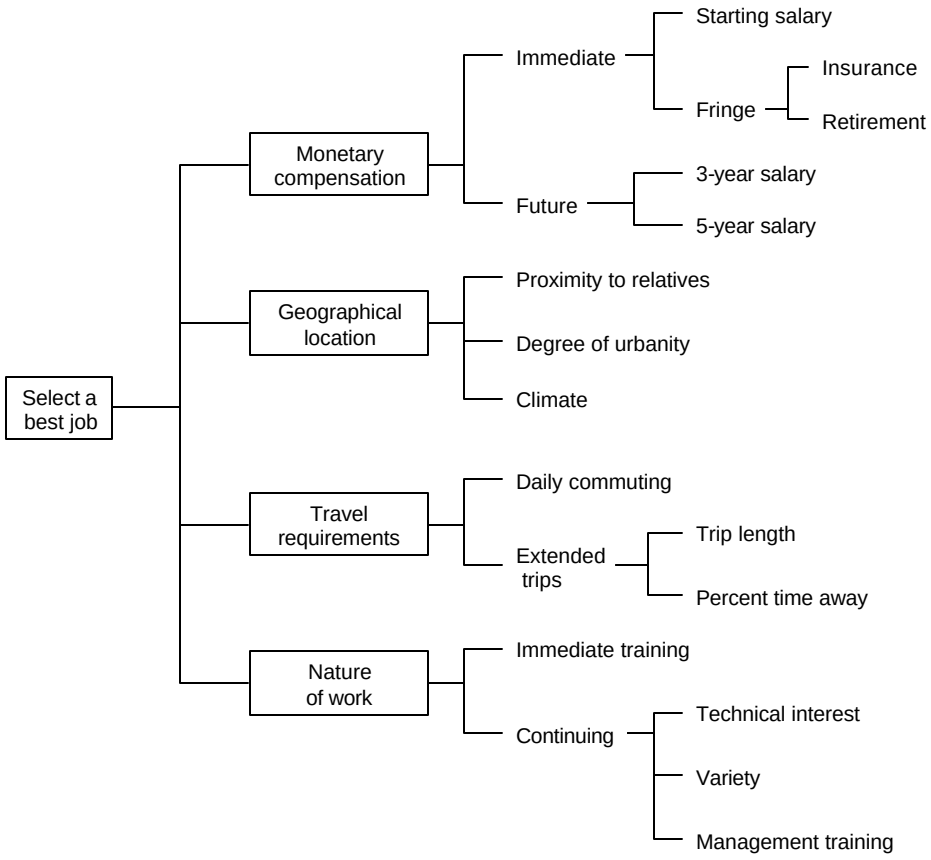


Figure 56: Value hierarchy for employment options [18]

The example in Figure 56 only shows the evaluation considerations. It is not generally practical to include the information regarding objectives and evaluation measures on the same diagram because it increases the complexity of the diagram. This additional information is typically captured in a table for reference. The evaluation considerations included in this table are those that are the leaves of the tree. Figure 57 is a partial evaluation consideration table for the job seeker example in Figure 56.

Evaluation Consideration	Objective	Evaluation Measure
Starting Salary	Maximize starting salary	Salary in current dollars
Insurance	Minimize health insurance premiums	Premium cost per year in current dollars
Retirement	Maximize retirement investment potential	Potential annual investment in current dollars
…	…	…

Figure 57: Evaluation considerations for evaluation of jobs

Value hierarchies are able to quickly convey information regarding a decision to the decision maker and other interested parties involved. When creating a value hierarchy, it is important to try to attain a set of desirable properties that will increase the value of the diagram. A value hierarchy that has these properties can provide many uses to the vested parties. The next sections discuss those properties and looks at the different uses a value hierarchy can meet.

A.6.2.2.1 Desirable Properties of Value Hierarchies

The desirable properties of value hierarchies are completeness, nonredundancy, decomposability, operability and small size [16, 18]. Each of these properties is discussed below.

A.6.2.2.1.1 Completeness

To be complete, the evaluation considerations at each layer in the hierarchy, as a group, must adequately cover all concerns in order to evaluate the overall objective of the decision. In addition to adequately covering the concerns, the evaluation measures at the lowest tier must adequately measure the degree of attainment for the respective objectives. A hierarchy that is complete is considered "collectively exhaustive." That is, taken as a whole, the value hierarchy includes everything necessary to evaluate the decision. Completeness is necessary to ensure that one can accurately distinguish the differences between alternatives. The task of developing evaluation measures is discussed in Section A.6.2.4.2.

A.6.2.2.1.2 Nonredundancy

As well as being complete, a value hierarchy should be nonredundant. In order to be nonredundant, evaluation considerations in the same tier should not overlap. For example, in Figure 56 the evaluation consideration "Immediate" under the "Monetary compensation" consideration is divided into "Starting salary" and "Fringe". Dollar values associated with these two evaluation considerations can be assigned to one, and only one, category; hence the categories are nonredundant. A nonredundant hierarchy is called "mutually exclusive." Factors used for evaluation are included in no more than one evaluation consideration.

A.6.2.2.1.3 Decomposability

Decomposability deals with the independence of evaluation measures for different evaluation considerations. When evaluation measures are not decomposable, developing a procedure to combine evaluation measures for determining an overall preferability of an alternative can be very difficult. In fact, the required procedures can become so complicated that they are no longer practical for use.

A.6.2.2.1.4 Operability

Operability of a value hierarchy implies understandability. The evaluation considerations, objectives and evaluation measures need to be understood by the audience. Technical terms and evaluation measures that are only understood by subject matter experts are impractical if people from different backgrounds are vested in the problem.

A.6.2.2.1.5 Small Size

The last property is size. There is benefit to having a smaller value hierarchy if it can communicate the same information as a larger hierarchy. The smaller value hierarchy is more easily explained to others and requires fewer resources in order to evaluate different alternatives. For practical value hierarchies, completeness and detail must be balanced with the ability to complete an analysis in a timely manner. To help keep the size reasonable, evaluation considerations should be put against a "test of importance" [18]. The test states, "an evaluation consideration should be included in a value hierarchy only if possible variations among the alternatives with respect to the proposed evaluation consideration could change the preferred alternative" [18].

A.6.2.2.2 Uses of Value Hierarchies

In addition to using a value hierarchy to evaluate alternatives, the value hierarchy has other important uses. First, the value hierarchy can guide information collection. By knowing what is important to the problem, appropriate evaluation measures can be selected or developed. At some point information collection must end and a decision made. Knowing what needs to be collected assists the decision maker in

knowing when enough information has been collected. Being able to have peers review the hierarchy and find holes helps avoid collecting too little information.

Another use of the value hierarchy is to help identify alternatives. If the situation does not call for a prescribed set of alternatives, the value hierarchy provides a basis for developing alternatives. On the other hand, some decision makers limit the number of alternatives that are considered, which could result in good alternatives being missed.

A third use of a value hierarchy is to facilitate communications between multiple stakeholders. The diagram can assist everyone in understanding what is collectively considered the important considerations, as well as provide a basis for compromise or consensus when selecting alternatives.

A.6.2.3 Evaluation Measures

Evaluation measures allow for a quantifiable degree of attainment of objectives. Quantifying evaluation considerations allow for an unambiguous rating on how well one alternative does compared to another with regard to a specified objective. Developing accurate evaluation measures is a critical part of the first step in the five-step decision making process. This section looks at the different types of evaluation measures and how to develop them.

A.6.2.4 Types of Evaluation Measure Scales

Evaluation measures are classified based on their occurrence in the world and how they measure objective attainment. That is, evaluation measures can occur naturally or be constructed for the particular decision. Additionally, evaluation measures can directly measure the attainment of the objective or reflect the degree of attainment.

A.6.2.4.1.1 Natural or Constructed

A *natural* scale for an evaluation measure is one that has a common interpretation for the general public. Natural scales have the property that not much time is required to define the scale, for example, a natural scale for the job seeker's "starting salary" evaluation measure, salary in current dollars, has the

intuitive definition that $50,000 is better than $25,000. There are situations, however, where a natural scale is not appropriate. In those cases, a *constructed* scale is used. A constructed scale is one that is developed for a particular problem. Constructed scales are necessary when natural scales do not exist for a particular evaluation measure.

A.6.2.4.1.2 Direct or Proxy

The second category for classifying evaluation measure scales specifies how the scale measures the degree of attainment. A *direct* scale, as the name implies, directly measures the degree of attainment. For example, in a business, "profits in dollars" is a direct scale. Scales that are a reflection of the degree of attainment are called *proxy* scales. The gross national product is a proxy scale for the economic well-being of a country.

A.6.2.4.2 Developing Evaluation Measure Scales

The purpose of developing an effective evaluation measure scale is to have an unambiguous rating of fitness one alternative does with a particular evaluation consideration compared to another alternative. The task of selecting scales may not be void of controversy. Depending on the type of scale used, interpretation of the scale by the interested parties may vary. The questions listed below are guides for selecting and developing evaluation measure scales for the decision problems evaluation considerations [18].

1. *Should we use a natural scale that is a proxy, or should we develop a constructed direct scale?*
2. *Should we subdivide an evaluation consideration into more detailed sub-considerations for which natural scales might exist, or should we construct a scale to measure the evaluation consideration without subdividing it further?*
3. *Should we use a natural scale that is precise, but uses technical jargon, or should we use a constructed scale that may not be as precise but that may be more understandable to some stakeholders in the decision process?*
4. *How carefully should we specify the scale definition for a constructed scale?*

A.6.2.5 Alternatives

The next step in the decision making process is developing viable alternatives to the decision problem. Even the most complete analysis can only show the best alternatives that have been identified. Likewise, if none of the alternatives are particularly good, the best solution the analysis can provide is the best of a poor set of alternatives. The value hierarchy can be used to assist decision makers in identifying alternatives, however, it is an explicit step and not a direct result from the analysis. There are many issues that come from having too many or too few alternatives; likewise, there are ways of identifying more potential alternatives. For additional information in this area, see [18].

A.6.2.6 Multiobjective Value Analysis

The purpose of multiobjective value analysis is to derive a value function that provides the fitness of an alternative to a particular problem. It is impossible to simultaneously maximize all objectives in a multiobjective decision problem [6]. The ability of a decision-maker to determine an optimal solution is based on defining an optimal solution to be "one that maximizes a decision maker's utility (or satisfaction)" [16]. This next section discusses how to derive the multiobjective value function to determine the decision maker's satisfaction and how to develop spreadsheet analysis techniques to validate the result.

A.6.2.6.1 Multiobjective Value Function

A multiobjective value function, or value function, combines the multiple evaluation measures of a decision problem into a single measurement of an evaluation alternative. The form of the value function is the weighted sum of the single dimensional value functions for each evaluation measure in the problem. Weights are applied to the results of single dimensional value functions in order to handle different scales in the functions as well as different levels of emphasis that are placed on each evaluation measure.

Generally when making a decision to a complex problem, a "winner" among a set of alternatives is not clear, as there are tradeoffs that occur between evaluation measures. By combining the values of the single dimensional value function into a single index value the overall desirability of an alternative can be

found. There are some intuitive combination techniques, like simple averaging, that are practical but have many drawbacks. Common problems include the units used for the evaluation measures. The scale used for a particular measure may cause the measure's value to dominate the value function. Likewise, ranges for the evaluation measures can pose problems. To solve these two potential problems, weights for each evaluation measure are introduced into the multiobjective value function.

These weights provide for a different multiplying factor for each evaluation measure to account for both the range of variation for each evaluation measure and different degrees of importance on each. That is, if one evaluation measure has a greater importance in the decision, it should have a greater weight.

Another problem that can arise is the return-to-scale issue. When using constructed evaluation measure scales, the difference between levels may not be equal. Take for example the following constructed evaluation measure scale for selecting a new software product:

-2	The use of the software will cause a potentially serious decrease in productivity
-1	There is a noticeable but acceptable decrease in productivity
0	There is no detectable change in productivity
1	There is an increase in work productivity

The movement from -2 to -1 may be significantly more important to the decision than the move from -1 to 0. When the increase in value that results from moving to each successively more preferable score on the scale is less, there is a decreasing return to scale. Likewise, when each successively more preferable score on the scale is greater than the last, there is an increasing return to scale. The returns-to-scale effect can be captured in the single dimensional value function.

The value function is of the following form

$$v(X_1, X_2, \ldots) = w_1 v_1(X_1) + w_2 v_2(X_2) + \ldots$$

where X_1, X_2, etc., are the appropriate evaluation measures; w_1, w_2, etc., are the weights on the evaluation measures; and $v_1(X_1)$, $v_2(X_2)$, etc., are the single dimensional value function for each of the evaluation

measures. The sum of the values of the weights typically sum to one. The single dimensional value functions are often piecewise linear or exponential functions.

A.6.2.6.2 Spreadsheet Analysis

The use of computer-based spreadsheets allows for quick calculation when making a decision. There are two major reasons for using a spreadsheet. First, it reduces the potential for human calculation errors. The second benefit is the ability to investigate whether changes to the assumptions about the data for the decision lead to changes in the ranking of alternatives. This is called sensitivity analysis. Since the sensitivity analysis can be tedious in the calculations, the spreadsheet makes the analysis fast and painless.

A.7 Summary

This appendix has provided a broad survey of many topics in software engineering with a focus on how they relate to software engineering paradigm selection. Beginning with some basic definitions, several properties of multiagent systems were explored. These properties included the multiagent system benefits, environments, principles and problem domains.

Next, three popular software engineering paradigms—object-oriented, component-ware, and agent-oriented—were discussed. Similarities to the approaches have been pointed out, as well as differences. Though object-oriented and agent-oriented software engineering appear to be more closely related than component-ware in technique, both the object- and agent-oriented approaches are appropriate for building system components, and the techniques used by the component-ware paradigm are reasonable for building agent systems in which the agents are heterogeneous.

Following the discussion of the three software engineering paradigms, a survey of problems and pitfalls that the software engineer may encounter when building agent-based systems was presented. From Section A.4, the unpredictability of agents and agent systems was described. Additionally, twenty-four pitfalls of the agent-oriented approach were summarized.

With the foundation of multiagent systems and software engineering techniques laid, several areas of related research were presented. Wooldridge and Jenning's work points out that the decision to use an agent-based approach is currently an open issue. The MESSAGE project is intent on defining guidelines

that software developers can use when making that decision. Finally, Software Engineering Institute research on selecting appropriate methodologies for developing real-time systems was discussed.

Last, techniques for strategic multidimensional decision-making are presented. Understanding of decision-making processes is an essential piece of this research. Decision theory allows for making a difficult decision easier to justify.

Appendix B. Methodology Selection Criteria Survey

B.1 Survey

Survey

Before beginning the survey, I am requesting some personal information. This information is being collected to track the background of the participants. Names and contact information are optional and will only be used in the event that we need any clarifications on the answers given.

Contact Information

Name:

E-Mail Address:

Background

Occupational Area:

Field of Interest (Multiagent Systems, Agent-Oriented Software Engineering, Methodologies, etc.):
Length of Time in Field:

Directions

Below are the sets of criteria for the Management and Technical Issues. For each of the issues, rate the categories on a scale of zero to four. (NR is provided as *not rated*). The rating that you are providing is based on the importance that the category has on the general problem. When evaluating a methodology for a specific software requirement the decision-maker will weight each of the factors.

Following each category is a brief description of the category as well as a field for any comments you may have regarding the category.
At the end of the survey, please remember to submit the results.

Management Issues

NR 0 1 2 3 4

Cost of Acquiring the Methodology

The category focuses on the costs involved with adopting the methodology for use. Factors that play into this category include the costs incurred by sending personnel to available training, the purchase of reference materials, etc.
Comment:

NR 0 1 2 3 4

Cost of Acquiring Support Tools

In addition to adopting a methodology, costs are incurred by acquiring tools that support the methodology. The tools include CASE tools that support the methodology and programming development tools. Additionally, this category factors in other software/hardware requirements that would be needed to support the tools.
Comment:

NR 0 1 2 3 4

Availability of Reusable Components

Incorporating previously developed software into a new system reduces the overall design, implementation, and testing phases for software development. This category is used to measure how easily the methodology allows the integrating predefined components into the system.
Comment:

NR 0 1 2 3 4

Support of Organizational Business Practices

The point of this category is to measure the impact the methodology will have on the Organizations Business Practices. These practices include tracking development progress through milestones, reports, customer interaction, etc.

Comment:

Compliance with Established Standards

NR 0 1 2 3 4

This category rates an alternative based on its ability to meet established standards, whether local or international.

Comment:

Support for Tracking Changes

NR 0 1 2 3 4

This category measures how well a methodology supports the ability to make and trace changes throughout the development lifecycle.

Comment:

Technical Issues

Integration of Existing Legacy Systems

NR 0 1 2 3 4

This category is used to measure the methodology's support for integrating the new system with related existing systems.

Comment:

Distribution

NR 0 1 2 3 4

The ability of the methodology to support the modeling of distributed aspects of the problem is the focus of this category.

Comment:

NR 0 1 2 3 4

Execution Environment

This category is used to determine the methodology's support of developing software systems for environments that have heterogeneous hardware or software (such as operating systems).

Comment:

System Structure

NR 0 1 2 3 4

This category measures the methodology's ability to develop software capable of handling the introduction and removal of system components in a manner that is not detrimental to the users of the system.

Comment:

Interactions

NR 0 1 2 3 4

This category determines the methodology's ability to handle the interaction between system level components as well as with entities outside of the system such as the human user or other systems.

Comment:

System Scalability

NR 0 1 2 3 4

This category is used to measure the methodology's ability to develop software capable of handling the introduction and removal of system-level resources while minimizing the impact on users.

Comment:

NR 0 1 2 3 4

Agility and Robustness

This category measures the methodology's ability to create flexible software that will be resilient to dynamic changes.

Comment:

Additional Questions

Would you include any additional factors? If yes, please list:

When making a decision based on these factors, do you believe the factors should be weighted relative to all of the other factors, or only relative to the factors within the same category of issues (Management or Technical)?

Yes No Unsure

If weighted relative only to the other factors in the category, what do you feel is an appropriate weighting for each category?

Management: %

Technical: %

Please check here if you would like a copy of the thesis this survey is supporting

Don't forget to submit your results!

B.2 Responses

Occupation: Government Time in Field: 1 - 5 years
Field: Interface agents, multiagent systems
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 2
Component Reuse: 2
Organizational Business Practices: 3
Compliance with Standards: 3
Effects of Changes: 3
Legacy System Integration: 3
Distribution: 1
Environment: 1
Dynamic Structure: 1
Interaction: 3
Don't ignore the simple fact that human users must interact with these systems
Scalability: 2
Agility and Robustness: 3
Additional Factors
Relative Weighting: UNSURE Management: Technical:

Occupation: Academic Time in Field: 1 - 5 years
Field: Multi-agent systems, real-time systems
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 3
Component Reuse: 4
Organizational Business Practices: 2
Compliance with Standards: 4
Effects of Changes: 3
Legacy System Integration: 1
Distribution: 3
Environment: 3
Dynamic Structure: 3
Interaction: 2
Scalability: 3
Agility and Robustness: 2
Additional Factors
Relative Weighting: UNSURE Management: 35 Technical: 65

Occupation: Industry/Commercial Time in Field: 5 - 10 years
Field: Multiagent Systems, logic programming, agent architectures, business models
Cost of Acquiring the Methodology: NR
This question is off base; especially as the first question.
Cost of Acquiring the Support Tools: 2
Ditto
Component Reuse: 2
Organizational Business Practices: 1
Compliance with Standards: 2
Effects of Changes: NR
Legacy System Integration: 3
Distribution: 0
Environment: 2
Dynamic Structure: 3

Interaction: 3
Scalability: 3
Agility and Robustness: 3
Additional Factors

This whole questionnaire begs the most important question: what do people see as the prime benefits and risks associated with agents? Why is a much more important question than how, or how much.

Relative Weighting: UNSURE Management: 75 Technical: 25

Occupation: Industry/Commercial Time in Field: 1 - 5 years
Field: Methodologies, Software Architecture, Design Patters, Components, Agents
Cost of Acquiring the Methodology: 2
Cost of Acquiring the Support Tools: 2
Component Reuse: 0
Organizational Business Practices: 1
Compliance with Standards: 3
Effects of Changes: 1
Legacy System Integration: 3
Distribution: 4
Environment: 1
Dynamic Structure: 1
Interaction: 2
Scalability: 3
Agility and Robustness: 3
Additional Factors
Relative Weighting: UNSURE Management: 40 Technical: 70

Occupation: Industry/Commercial Time in Field: 5 - 10 years
Field: Multiagent Systems as applied to Enterprise computing problems
Cost of Acquiring the Methodology: 1

Cost is not as important as the cost/benefit ratio. If my benefit is measured in the millions, it matters not if my entry cost is $10,000 or $100,000. (Cost/benefit is perhaps scored as a 4.)

Cost of Acquiring the Support Tools: 1

same as above

Component Reuse: 2

We tend to be COTS oriented. ...but we are learning fast that it is more important to have things work right than to save a few bucks by applying the shoehorn.

Organizational Business Practices: 2
Compliance with Standards: 4

Boeing is big. As such, interoperability is extremely important.

Effects of Changes: 3

Our complexity and scale requires strong configuration management and the use of block points.

Legacy System Integration: 4
Distribution: 2
Environment: 4
Dynamic Structure: 4
Interaction: 3
Scalability: NR

This question is confusing. To me scalability (in the title) has to do with sheer numbers ... high volumes of transactions, large numbers of users, etc without bogging down. The description talks more about changes and disruption. Agreed, disruption on a large scale

can have a larger overall impact than disruption on a smaller scale, but scalability and disruption are two very different things. As for my score, it all depends on the application and my first applications will be smaller prototype apps where adverse impacts are minimized.

Agility and Robustness: 3

Additional Factors

Benefits, cost savings, and productivity gains are also important ...but in the context of the application. As I look at this questionaire, I get the sense that this is the criteria for selecting a vendor's package. We do not divorce the two. The weight of various criteria depends on how we might use the product or tool.

Relative Weighting: UNSURE Management: Technical:

Occupation: Industry/Commercial Time in Field: > 10 years

Field: Intelligent Agents, Engineering Process Automation

Cost of Acquiring the Methodology: 1

The organization within EDS I am involved in is standardizing on the CMM/SEI methodologies, as long as this can tie directly to that, it would make it much easier for the industry to digest...

Cost of Acquiring the Support Tools: 2

Component Reuse: 3

Some of the things I have in mind with Agent Technology requires the ability to utilize installed software (existing 3rd Party Engineering Analysis software)

Organizational Business Practices: 3

Again, my organization has requirements in place. We are attempting to get our SEI Level 4 certification, so our methodologies are pretty well ingrained...

Compliance with Standards: 4

As mentioned earlier, we are kind of committed to specific standards already...(As a company with over 120,000 employees, it is slow to move)

Effects of Changes: 3

This is something that occurs with methodologies already and must be maintained...

Legacy System Integration: 1

To prove feasibility, early systems will be self sufficient, as the trust in the technology increases so will the level of integration...

Distribution: 3

Our focus is global, we have vehicle development activities all over the world that need to work together...

Environment: 4

The things I am thinking about require multi-platform capability. We have groups of users on UNIX (HP and Sun), Windows (95 and NT), and soon to be 2000...

Dynamic Structure: 3

Minimize impact on the user as much as possible when doing enhancments...

Interaction: 4

Depending on the ability of the technology, many different systems could be users or suppliers...

Scalability: 3

As mentioned before, as trust increases so will it's use...Users also tend to find uses you never dreamed of...

Agility and Robustness: 3

This is one of the reasons I am thinking about this technology...

Additional Factors

As with other new technology and concepts, specific examples of true capabilities and restrictions or limitations would be extremely helpful... One of the things I struggle with most is how to apply these concepts to my specific needs in the Engineering (at least Automotive development) environment...

Relative Weighting: YES Management: 35 Technical: 65

Occupation: Academic Time in Field: 5 - 10 years
Field: Multiagent Systems, AOSE, Artificial Life
Cost of Acquiring the Methodology: 2
Cost of Acquiring the Support Tools: 3
Component Reuse: 3
Organizational Business Practices: 1
Compliance with Standards: 1
Effects of Changes: 4
Legacy System Integration: 1
Distribution: 2
Environment: 3
Dynamic Structure: 2
Interaction: 3
Scalability: 4
Agility and Robustness: 4
Additional Factors
Relative Weighting: UNSURE Management: 35 Technical: 65

Occupation: Industry/Commercial Time in Field: 5 - 10 years
Field: Multiagent systems
Cost of Acquiring the Methodology: 3

All anwswers are in the context of the Open Agent Architecture (OAA), www.openagent.com, a multiagent framework. OAA puts forth a new style of programming called "Delegated Computing" that is radically different enough as to create somewhat of a barrier until the programmer learns to trust the methodology.

Cost of Acquiring the Support Tools: 0

No support tools required, so cost is low.

Component Reuse: 4

Highest reuse of components I've seen.

Organizational Business Practices: 2

Should be as good as other methodologies.

Compliance with Standards: 1

OAA does not expend much effort to support other standards.

Effects of Changes: 1

No additional support beyond standard programming tools.

Legacy System Integration: 4

Support for many programming languages, so it's easy to write adapters for legacy systems.

Distribution: 4

Highly distributed.

Environment: 4

OAA supports many platforms, OS, (Solaris, Windows, Linux) and programming languages.

Dynamic Structure: 4

Completely dynamic, any component can be added or removed at runtime and the system degrades in as optimal a way as possible.

Interaction: 4

Unparalleled capabilities for enabling cooperation and advanced interactions among multiple human and automated components.

Scalability: 2

Most multiagent systems are only research quality, as is OAA. Robust, but only moderately scalable (~50 agent components or so).

Agility and Robustness: 4
None better.
Additional Factors
Relative Weighting: UNSURE Management: Technical:

Occupation: Government Time in Field: 5 - 10 years
Field: human interaction with agent-based systems
Cost of Acquiring the Methodology: 4
Cost of Acquiring the Support Tools: 4
Component Reuse: 3
Organizational Business Practices: 4
Compliance with Standards: 2
Effects of Changes: 4
Legacy System Integration: 4
Distribution: 4
Environment: 4
Dynamic Structure: 3
Interaction: 4
Scalability: 4
Agility and Robustness: 4
Additional Factors
No additional factors, but I question how useful the survey results will be. I would say that all the factors above are critical. You might want to think about reworking this into a relative weighting survey. I.e., 16 factors, rated in order of importance. Otherwise I'm afraid you're going to find some surveys where everything gets a '4'. Good luck. I don't want an individual copy of your thesis, but please be sure to advertise it on the agents-list when it is completed.
Relative Weighting: NO Management: Technical:

Occupation: Academic Time in Field: 1 - 5 years
Field: Software Engineering, MAS and AOSE
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 3
Also the training for these tools. In using every tool in our company we are focused on the training needed for that tools. Also, the level of company process maturity.
Component Reuse: 3
Organizational Business Practices: 4
Compliance with Standards: 2
It's needed for Quality Control of software artifacts that are produced during the methodology.
Effects of Changes: 3
Legacy System Integration: 2
Distribution: 2
Environment: 2
Dynamic Structure: 3
Interaction: 4
Scalability: 4
Agility and Robustness: 4
Additional Factors
Relative Weighting: YES Management: 45 Technical: 55

Occupation: Student Time in Field: 1 - 5 years
Field: Community information exchange support (agents as software development paradigm)
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 2

Component Reuse: 2
Organizational Business Practices: 1
Compliance with Standards: 2
Effects of Changes: 3
Legacy System Integration: 3
Distribution: 4
Environment: 4
Dynamic Structure: 4
Interaction: 2
Scalability: 4
Agility and Robustness: 3
Additional Factors
Relative Weighting: UNSURE Management: 75 Technical: 25

Occupation: Industry/Commercial Time in Field: 5 - 10 years
Field: Multiagent Systems, Mobile agents technology,
Cost of Acquiring the Methodology: 3

Comparing to other area filed of Agent technology is rather young. Material and information available are mostly results of research projects. For the commercial sector, time and money is needed to find out how this technology could be used for public use.

Cost of Acquiring the Support Tools: 2

There are lots of scientific products. Commercial tools are starting to become more available and mature

Component Reuse: 4

To my opinion this playes an important role. Reusable components would help to design and implement agent applications faster and cheaper. also complies with OO-paradigm that plays an important role now adays.

Organizational Business Practices: 3
Compliance with Standards: 4
Effects of Changes: 2
Legacy System Integration: 2
Distribution: NR
Environment: 1
Dynamic Structure: 2
Interaction: 4
Scalability: 4
Agility and Robustness: 2
Additional Factors
Relative Weighting: NO Management: 40 Technical: 60

Occupation: Industry/Commercial Time in Field: 5 - 10 years
Field: Agent-Oriented Software Engineering, Multiagent
Cost of Acquiring the Methodology: 3

There aren't many professional/commercial-oriented resources for personnel training.

Cost of Acquiring the Support Tools: 0

There are a lot of tools that you can use without many costs. The problem is with the guarantees you have with those tools. Most of them are academic or semi-academic.

Component Reuse: 1

Naturally, it depends on the previously used platform. In a Intranet base solutin the multi-agent interaction is quite simple.

Organizational Business Practices: 4

It will be a "revolution" in the way we interact with the information systems. The organizations will have to make some adjustments in their business process.

Compliance with Standards: 4

Effects of Changes: 3
Legacy System Integration: 3
Identical to Availability of Reusable Components
Distribution: 3
Multiagent systems are well fitted to local business analis ys.
Environment: 2
This is a dificult question. It depends on the development tools you choose. I think that there are already some mobile agents that are able to travel across networks of heterogeneous
Dynamic Structure: 4
Multiagent systems allow for on-running reconfiguration. But, naturally, it depends on how you implement the delegation and negotiation process.
Interaction: 3
The interaction with the users may be better than non-agent systems. The interaction with other systems may require some intermediate level (an interface agent?), which will degrade performance.
Scalability: 4
As stated before, multiagent systems are very flexible.
Agility and Robustness: 2
You can achieve systems that are quite robust against system break-down, but you may also get some unexpected system behaviour.
Additional Factors
Relative Weighting: NO Management: 60 Technical: 40

Occupation: Industry/Commercial Time in Field: 1 - 5 years
Field: MAS, AOSE, SE in General
Cost of Acquiring the Methodology: 1
Cost of Acquiring the Support Tools: 1
Component Reuse: 3
Organizational Business Practices: 2
Compliance with Standards: 3
Effects of Changes: 3
Legacy System Integration: 3
Distribution: 3
Environment: 1
Dynamic Structure: 0
Interaction: 1
Scalability: 3
Agility and Robustness: 4
Additional Factors
Relative Weighting: YES Management: Technical:

Occupation: Academic Time in Field: 1 - 5 years
Field: Agent Communication Languages, Multiagent Systems
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 2
Component Reuse: 4
Organizational Business Practices: 2
Compliance with Standards: 3
Effects of Changes: 2
Legacy System Integration: 4
Distribution: 4
Environment: 4
Dynamic Structure: 2

Interaction: 2
Scalability: 1
Agility and Robustness: 3
Additional Factors
Relative Weighting: UNSURE Management: Technical:

Occupation: Academic Time in Field: 1 - 5 years
Field: Agent-Oriented Software Engineering, Distance
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 2
Component Reuse: NR
Organizational Business Practices: 3
Compliance with Standards: 3
Effects of Changes: 4
Legacy System Integration: 3
Distribution: 4
Environment: 2
Dynamic Structure: 3
Interaction: 3
Scalability: 4
Agility and Robustness: 4
Additional Factors
Workflow technology, Human factors
Relative Weighting: YES Management: 60 Technical: 40

Occupation: Academic Time in Field: 1 - 5 years
Field: Multi-Agent Systems / Distributed agent operating
Cost of Acquiring the Methodology: 4
Cost of Acquiring the Support Tools: 4
Component Reuse: 4
Organizational Business Practices: 3
Compliance with Standards: 4
Effects of Changes: 4
Legacy System Integration: 4
Distribution: 4
Environment: 4
Dynamic Structure: 4
Interaction: 4
Scalability: 4
Agility and Robustness: 4
Additional Factors
What do you learn from this questionnaire? All aspects are important. Maybe give combinations of aspects, and ask for which one is more important than the other?! Also allows you to cross-check answers from people. The question below has an OR in it - hard to answer!
Relative Weighting: YES Management: Technical:

Occupation: Academic Time in Field: 1 - 5 years
Field: Multiagent Systems
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 4
Component Reuse: 2
Organizational Business Practices: 3
Compliance with Standards: 2
Effects of Changes: 2

Legacy System Integration: 2
Distribution: 4
Environment: 4
Dynamic Structure: 2
Interaction: 3
Scalability: NR
Agility and Robustness: 4
Additional Factors
Relative Weighting: UNSURE Management: 10 Technical: 90

Occupation: Student Time in Field: 1 - 5 years
Field: Infrastructures for Multi-Agent Systems
Cost of Acquiring the Methodology: 1
Cost of Acquiring the Support Tools: 2
Component Reuse: 3
Organizational Business Practices: 4
Compliance with Standards: 3
Effects of Changes: 4
Legacy System Integration: 4
Distribution: 2
Environment: 4
Dynamic Structure: 4
Interaction: 4
Scalability: 4
Agility and Robustness: 4
Additional Factors

The questions asked have a high "it depends" factor, especially for the technical section. Without any clear knowledge of the situation for which the technology is used the weight to all the questions is inevitably 4 since the questions represent the very essence of multi-agent systems ~(adaptability, sociability, reactiveness, interaction with users, integration of legacy systems etc, etc.). I don't know to which extend this will bias your survey since I find it difficult to believe that most people will not rate all aspects a s very important.
In any case, well done for performing the survey and I hope people participate.

Relative Weighting: YES Management: 30 Technical: 70

Occupation: Student Time in Field: 1 - 5 years
Field: Multi-agent Systems, Systems Engineering
Cost of Acquiring the Methodology: 1
Cost of Acquiring the Support Tools: 3
Component Reuse: 4
Organizational Business Practices: 0
Compliance with Standards: 0
Effects of Changes: 2
Legacy System Integration: 2
Distribution: 4
Environment: 3
Dynamic Structure: 4
Interaction: 4
Scalability: 3
Agility and Robustness: 4
Additional Factors
Relative Weighting: NO Management: 25 Technical: 75

Occupation: Academic Time in Field: > 10 years
Field: Agent-Oriented Software Engineering

Cost of Acquiring the Methodology: 4
Cost of Acquiring the Support Tools: 4
Component Reuse: 4
Organizational Business Practices: 4
Compliance with Standards: 4
Effects of Changes: 4
Legacy System Integration: 4
Distribution: 4
Environment: 4
Dynamic Structure: 4
Interaction: 4
Scalability: 4
Agility and Robustness: 4
Additional Factors
Expected Agent Lifetime
Relative Weighting: YES Management: 50 Technical: 50

Occupation: Academic Time in Field: > 10 years
Field: Software Engineering
Cost of Acquiring the Methodology: 1
If it's good enough, cost can usually be overcome.
Cost of Acquiring the Support Tools: 1
Component Reuse: 2
Although I think reuse is important, reality is that there aren't sufficient libraries available to make it practical in all but a few limited cases.
Organizational Business Practices: 4
The less I have to change established practices, the better.
Compliance with Standards: 2
It depends. Some standards aren't very good, but without them it's difficult to exchange models for reuse
Effects of Changes: 2
This is important, but I consider it separate from the development methodology
Legacy System Integration: 3
This is important, but again, I'm pessimistic about the ability for any methodology to support this. Thus, I would be hesitant to base my selection of a methodology based on this criteria
Distribution: 2
Certainly important if I'm building a distributed system
Environment: 2
Not sure I understand this one.
Dynamic Structure: 4
I like this.
Interaction: 3
In what way do you mean?
Scalability: 3
Sounds a lot like system structure
Agility and Robustness: 4
Important, but how do you measure it?
Additional Factors
Relative Weighting: YES Management: 30 Technical: 70

Occupation: Student Time in Field: 1 - 5 years
Field: Multiagent Systems,
Cost of Acquiring the Methodology: 0

Cost of Acquiring the Support Tools: 0
Component Reuse: 4
Organizational Business Practices: 2
Compliance with Standards: 2
Effects of Changes: 3
Legacy System Integration: 4
Distribution: 3
Environment: 3
Dynamic Structure: 3
Interaction: 4
Scalability: 1
Agility and Robustness: 4
Additional Factors
Relative Weighting: UNSURE Management: 10 Technical: 90

Occupation: Academic Time in Field: 1 - 5 years
Field: multi-agent systems
Cost of Acquiring the Methodology: 2
Cost of Acquiring the Support Tools: 2
Component Reuse: 3
Organizational Business Practices: 2
Compliance with Standards: 3
Effects of Changes: 3
Legacy System Integration: 2
Distribution: 2
Environment: 3
User interface important because trust plays a part
Dynamic Structure: 2
Interaction: 3
Scalability: 2
Agility and Robustness: 3
Additional Factors
Relative Weighting: UNSURE Management: Technical:

Occupation: Academic Time in Field: 1 - 5 years
Field: AOSE, OOSE, MAS
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 2
Component Reuse: 3
Organizational Business Practices: 3
Compliance with Standards: 2
Effects of Changes: 1
Legacy System Integration: 3
Do the systems being developed really need to be compatible with legacy systems.
Distribution: 1
How often is this necessary or important?
Environment: 0
How does a particular methodology determine or limit what kind of environments the systems being developed can operate on?
Dynamic Structure: 3
For any kind of upgradability, this would seem important.
Interaction: 1
Of course, this highly depends on what kinds of systems are going to be developed.
Scalability: 3

Agility and Robustness: 3
Additional Factors
Relative Weighting: UNSURE Management: 40 Technical: 60

Occupation: Student Time in Field: > 10 years
Field: Knowledge Based Software Engineering
Cost of Acquiring the Methodology: NR
N/A - as a student, I do not have a cost consideration
Cost of Acquiring the Support Tools: NR
N/A - as a student, I do not have a cost consideration
Component Reuse: 4
Integrating "provibly correct" software solutions to the generation of new software systems in support of the principle of software reuse is of great importance.
Organizational Business Practices: 1
Compliance with Standards: 1
Effects of Changes: 3
This is more important for large scale long term business or mission critical systems were many people are responsible of system maintenance.
Legacy System Integration: 2
Distribution: 2
Environment: 1
Dynamic Structure: 1
However, could be rated very high (3 or 4) if the reliablity concerns of the system were extremely important.
Interaction: 1
Scalability: 2
Agility and Robustness: 1
Additional Factors
Depending upon the category of the individual, the responses will vary greatly. As a student, I can only guess at the many very important commercial aspects of software systems (reliability and maintainability being in my option of great concern).
Relative Weighting: NO Management: 75 Technical: 25

Occupation: Academic Time in Field: 5 - 10 years
Field: Design of Intelligent multi-agent systems, automated negotiation, interpretation and verification of multi-agent systems, and more.
Cost of Acquiring the Methodology: 2
Cost of Acquiring the Support Tools: 2
Component Reuse: 4
Organizational Business Practices: 3
Compliance with Standards: 4
Effects of Changes: 2
Legacy System Integration: 1
Distribution: 4
Environment: 4
Dynamic Structure: 4
Interaction: 2
Scalability: 4
Agility and Robustness: 4
Additional Factors
- does the development method support compositional design
- does the method support validation and verification
- does the method support requirement engineering
Relative Weighting: YES Management: Technical:

Occupation: Academic Time in Field: > 10 years
Field: Multiagent Systems
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 2
Component Reuse: 2
Organizational Business Practices: 3
Compliance with Standards: 2
Effects of Changes: 1
Legacy System Integration: 3
Distribution: 4
Environment: 2
Dynamic Structure: 3
Interaction: 3
Scalability: 2
Agility and Robustness: 4
Additional Factors
Relative Weighting: UNSURE Management: Technical:

Occupation: Academic Time in Field: > 10 years
Field: Process activities and measurement
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 2
Component Reuse: 2
Organizational Business Practices: 3
Compliance with Standards: 2
Effects of Changes: 3
Legacy System Integration: 3
Distribution: 1
Environment: 2
Dynamic Structure: 3
Interaction: 2
Scalability: 4
Agility and Robustness: 4
Additional Factors
Question below is not a yes-no question. I assume yes means first choice.
Relative Weighting: YES Management: 45 Technical: 55

Occupation: Academic Time in Field: 1 - 5 years
Field: Juridical applications
Cost of Acquiring the Methodology: 4
We have only a very small budget, and are mostly interested in producing working prototypes to demonstrate our ideas.
Cost of Acquiring the Support Tools: 4
Component Reuse: 1
Organizational Business Practices: 0
Compliance with Standards: 2
Effects of Changes: 2
Legacy System Integration: 0
Distribution: 3
Environment: 2
Dynamic Structure: 2
Interaction: 3
Scalability: 1
Agility and Robustness: 3

Additional Factors
Relative Weighting: YES Management: 50 Technical: 50

Occupation: Government Time in Field: 5 - 10 years
Field: Information Technology, Computer Networks, Information Warfare
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 3
Component Reuse: 2
Organizational Business Practices: 3
Compliance with Standards: 4
Effects of Changes: 3
Legacy System Integration: 4
Distribution: 3
Environment: 3
Dynamic Structure: 2
Interaction: 2
Scalability: 2
Agility and Robustness: 3
Additional Factors
Relative Weighting: UNSURE Management: Technical:

Occupation: Academic Time in Field: > 10 years
Field: Formal methods
Cost of Acquiring the Methodology: 2
Cost of Acquiring the Support Tools: 2

This one is a little problematic. If tools are reasonable, then importance is 2. However, many CASE tools are prohibitively expensive. In that case cost would be a 4 as it could single-handedly eliminate the option.

Component Reuse: 3
Organizational Business Practices: 2
Compliance with Standards: 4
Effects of Changes: 4
Legacy System Integration: 3
Distribution: 2

This is very application-specific. For some (distributed) applications this is a 4.

Environment: 2

Unclear. Does a high rating mean it needs to run in a heterogeneous environment? Like the previous question, this is application specific.

Dynamic Structure: 2

Dynamically? Again, depends on the application.

Interaction: 4
Scalability: 3

Sounds similar to System Structure. Usually this means the system can grow (in some sense) with a linear impact on performance time.

Agility and Robustness: 4
Additional Factors

Availability of tools (not just cost).

Experience base (is this a new methodology or has it been proven in practice).

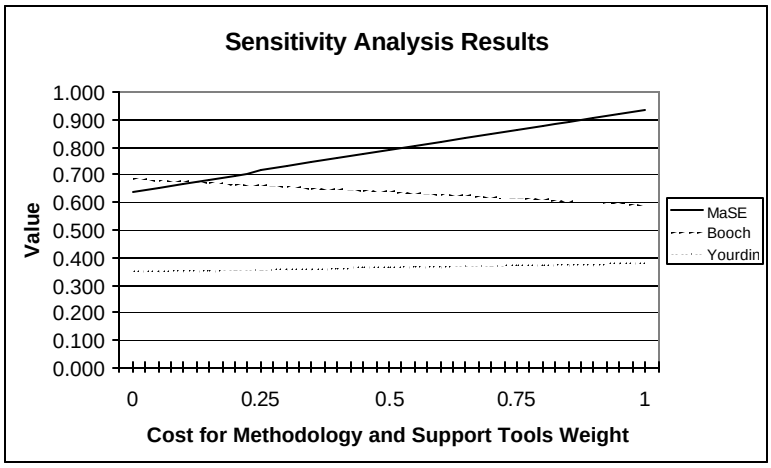
Relative Weighting: UNSURE Management: 33 Technical: 66

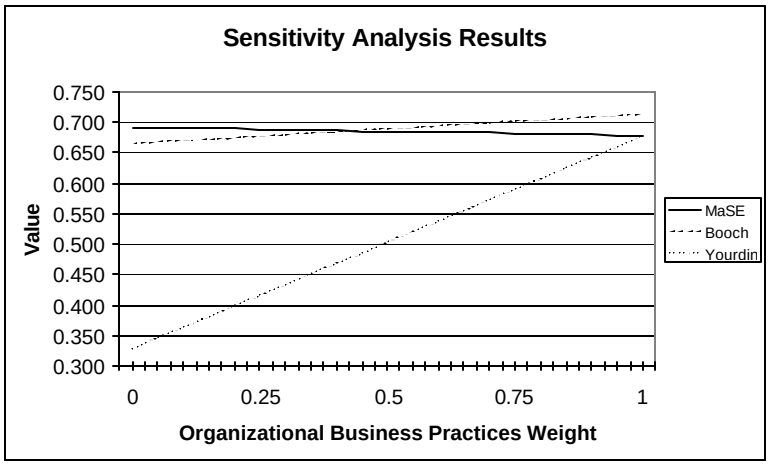
Occupation: Academic Time in Field: 1 - 5 years
Field: MAS, AOSE
Cost of Acquiring the Methodology: 3
Cost of Acquiring the Support Tools: 3
Component Reuse: 2

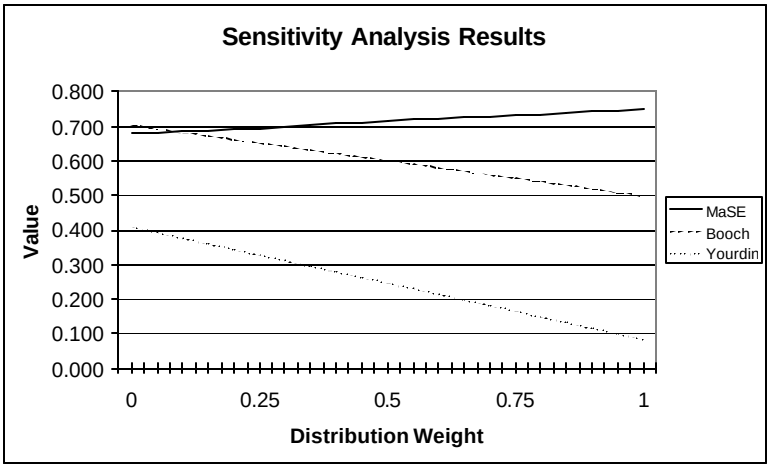
Organizational Business Practices: 4
Compliance with Standards: 4
Effects of Changes: 4
Legacy System Integration: 3
Distribution: 4
Environment: 3
Dynamic Structure: 4
Interaction: 4
Scalability: 3
Agility and Robustness: 3
Additional Factors
Relative Weighting: YES Management: 50 Technical: 50

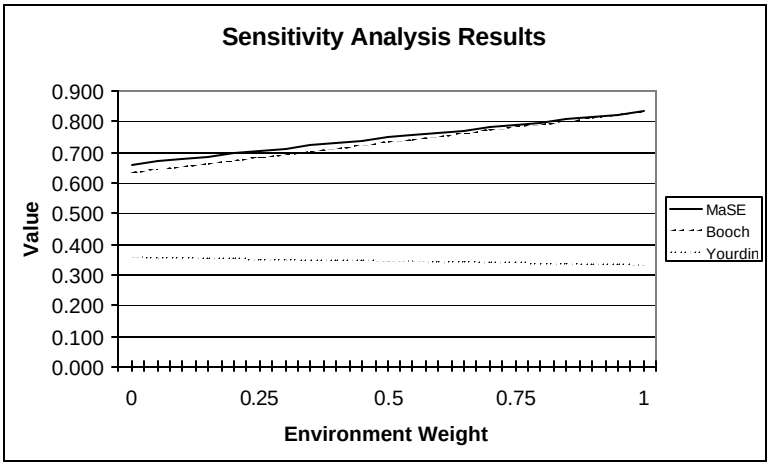
Appendix C. Sensitivity Analyses

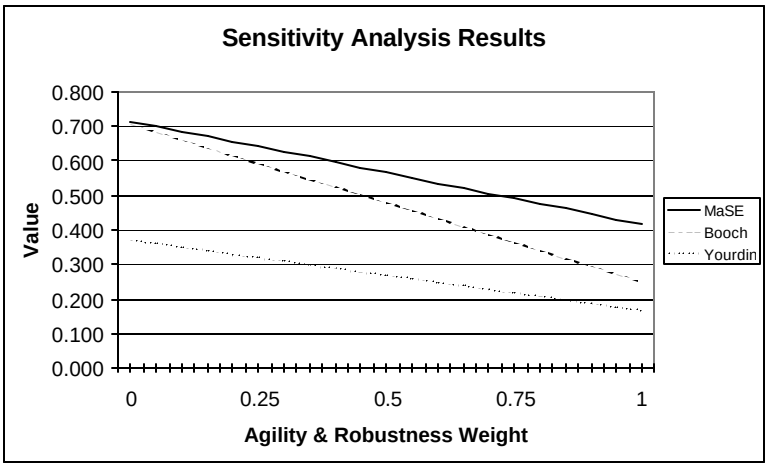
C.1 Case Study 1

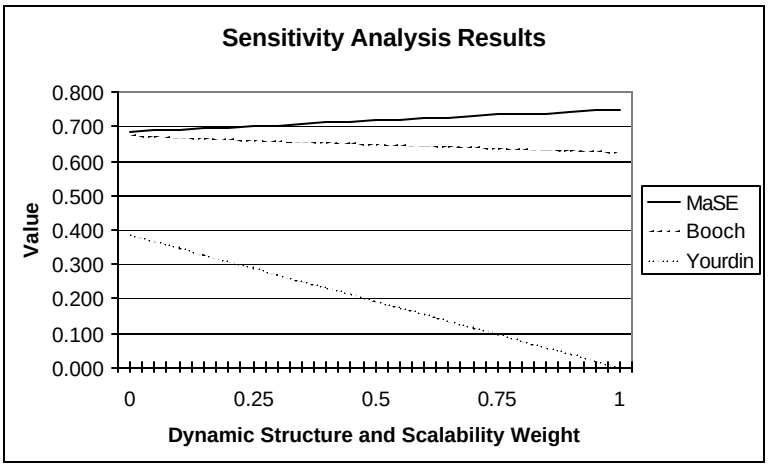


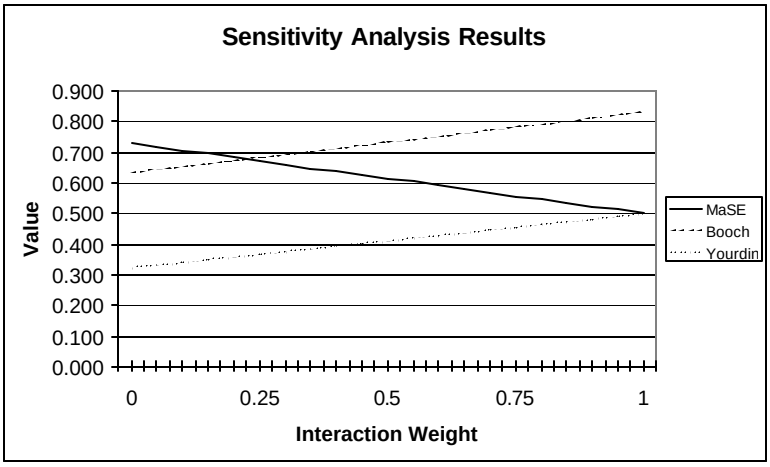


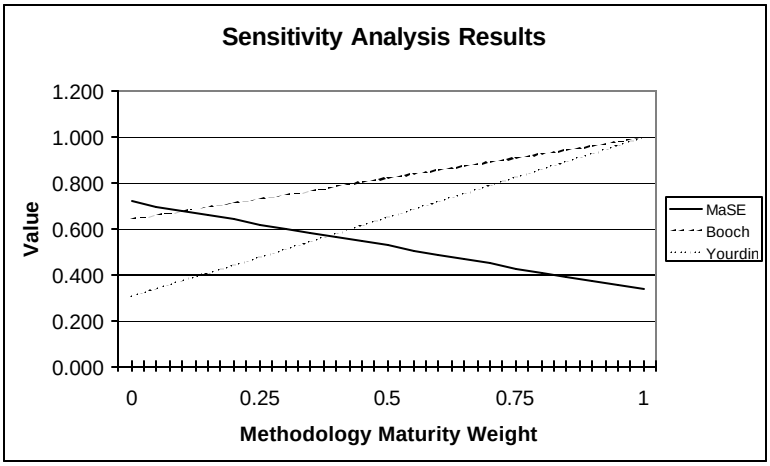

Sensitivity Analysis Results
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Distribution Weight
MaSE
Booch
Yourdin


Sensitivity Analysis Results
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Environment Weight
MaSE
Booch
Yourdin

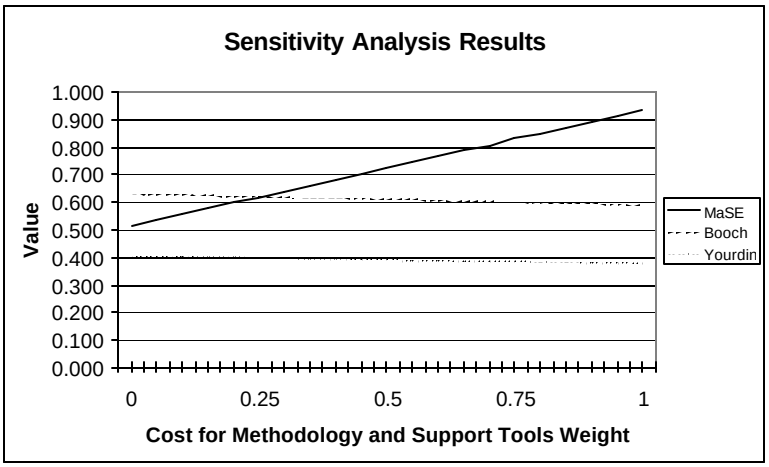

Sensitivity Analysis Results
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Agility & Robustness Weight
MaSE
Booch
Yourdin

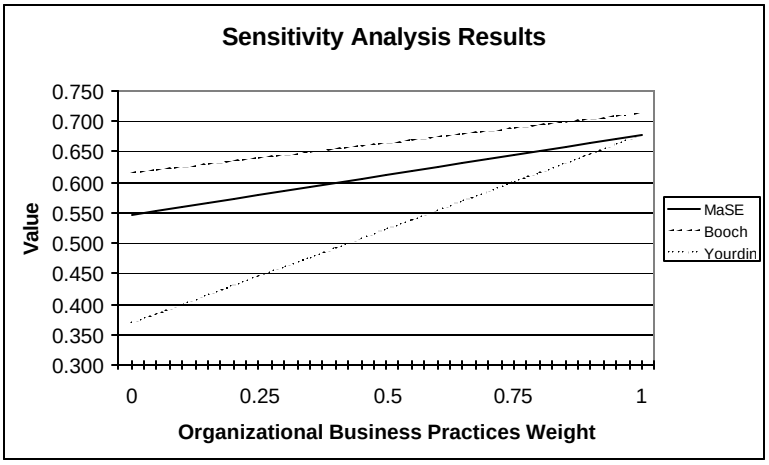

Sensitivity Analysis Results
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Dynamic Structure and Scalability Weight
MaSE
Booch
Yourdin

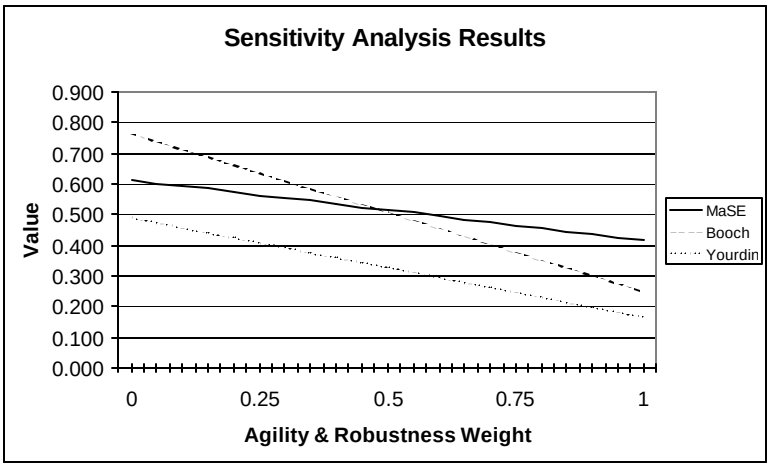

Sensitivity Analysis Results
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Interaction Weight
MaSE
Booch
Yourdin

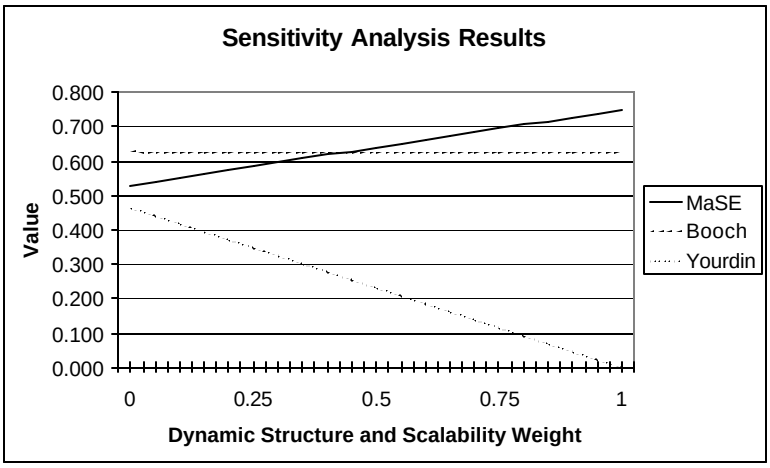

Sensitivity Analysis Results
1.200
1.000
0.800
0.600
0.400
0.200
0.000
Value
0
0.25
0.5
0.75
1
Methodology Maturity Weight
MaSE
Booch
Yourdin

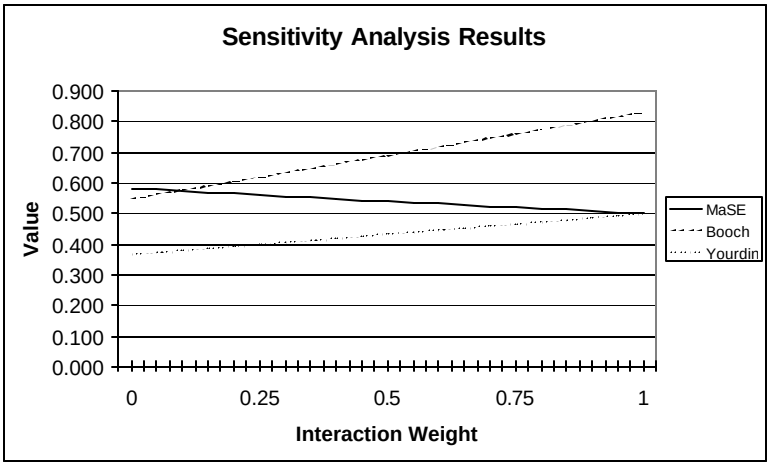
C.2 Case Study 2

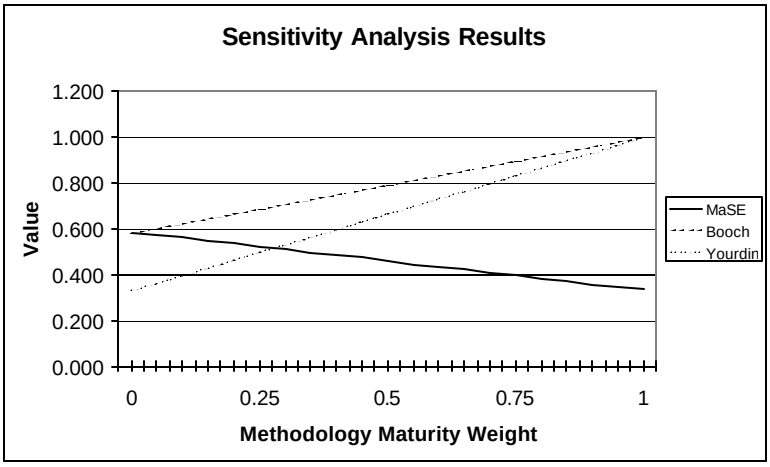




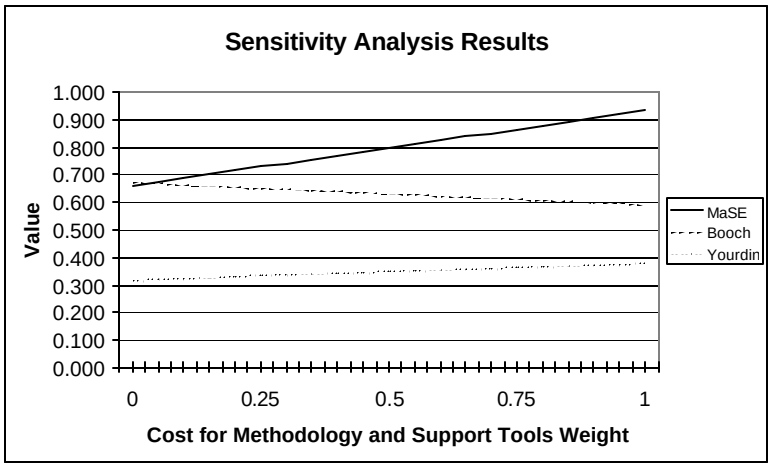


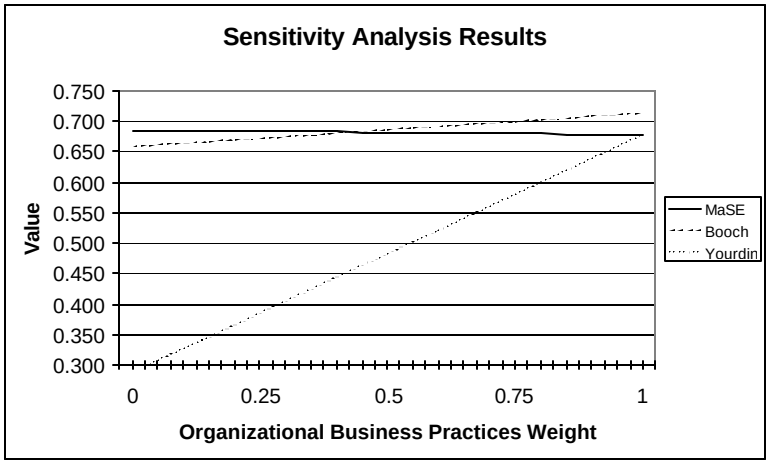

Sensitivity Analysis Results
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Dynamic Structure and Scalability Weight
MaSE
Booch
Yourdin

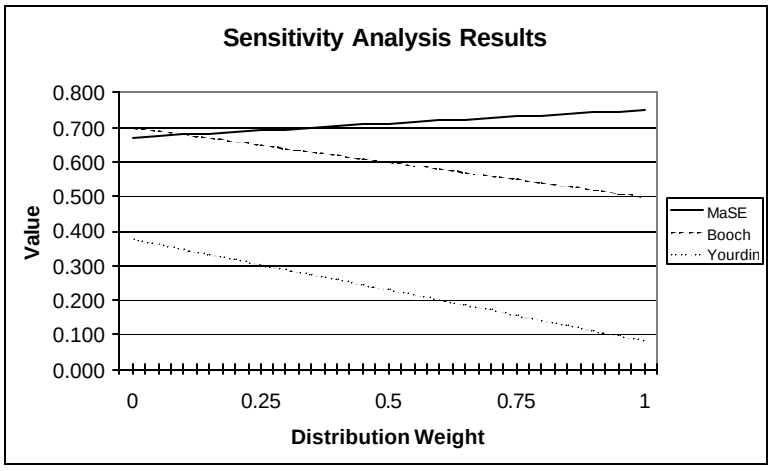

Sensitivity Analysis Results
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Interaction Weight
MaSE
Booch
Yourdin

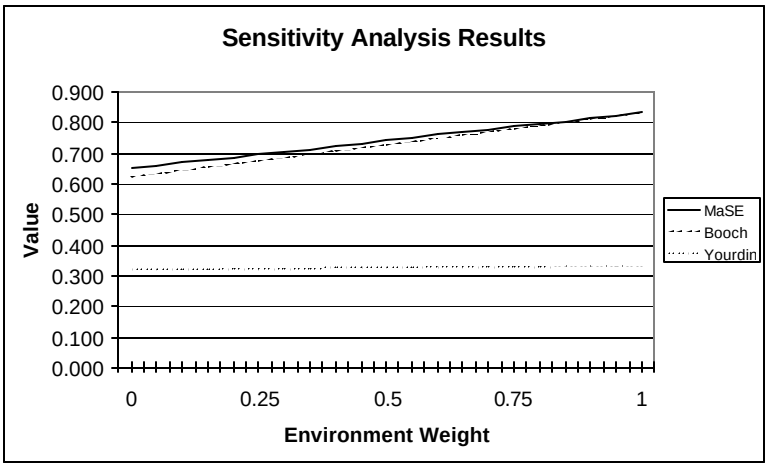

Sensitivity Analysis Results
1.200
1.000
0.800
0.600
0.400
0.200
0.000
Value
0
0.25
0.5
0.75
1
Methodology Maturity Weight
MaSE
Booch
Yourdin

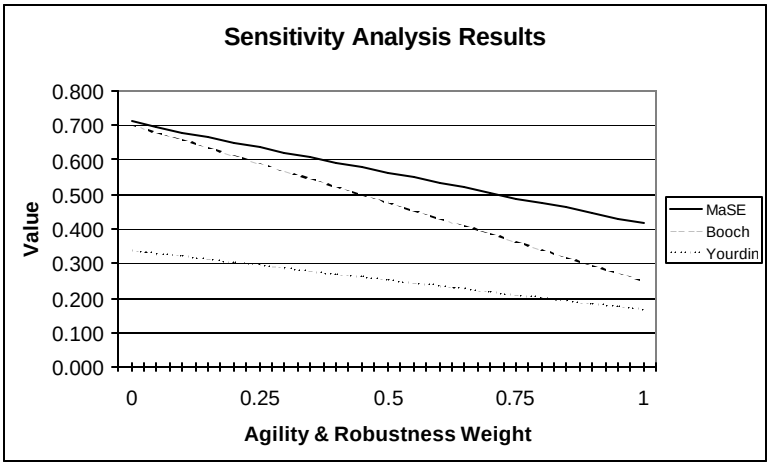
C.3 Case Study 3

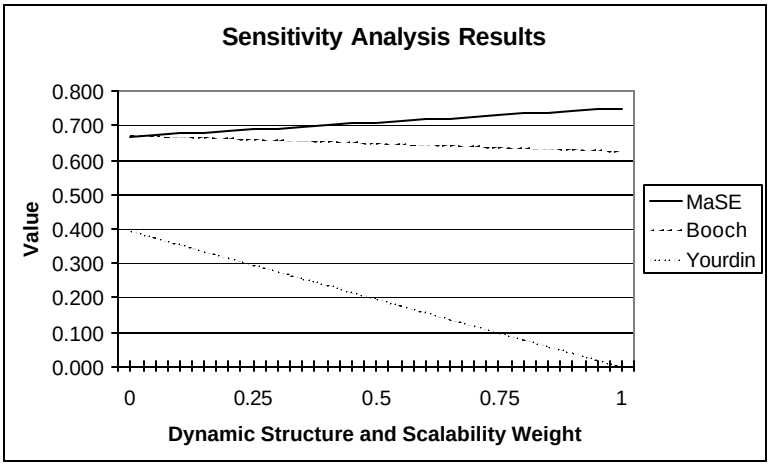


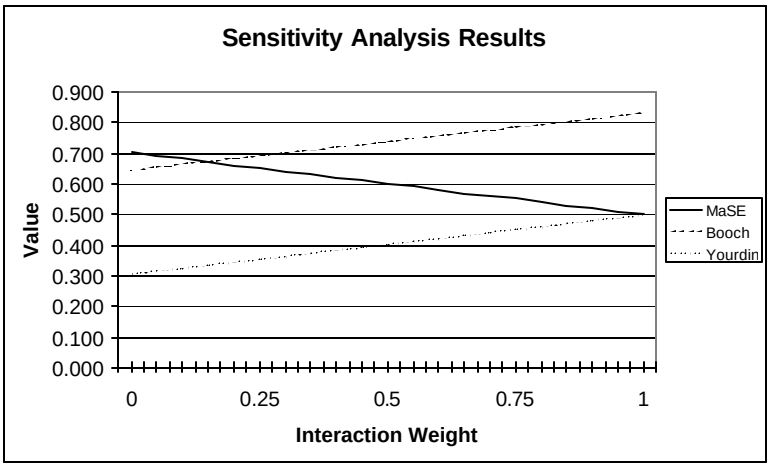


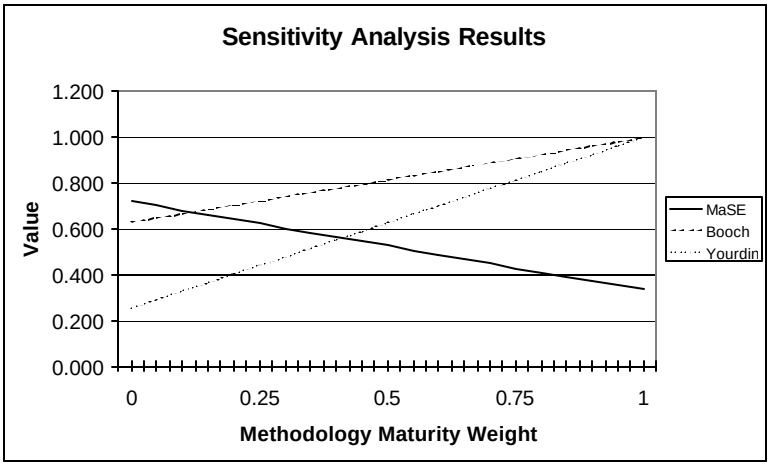



Sensitivity Analysis Results
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Environment Weight
MaSE
Booch
Yourdin

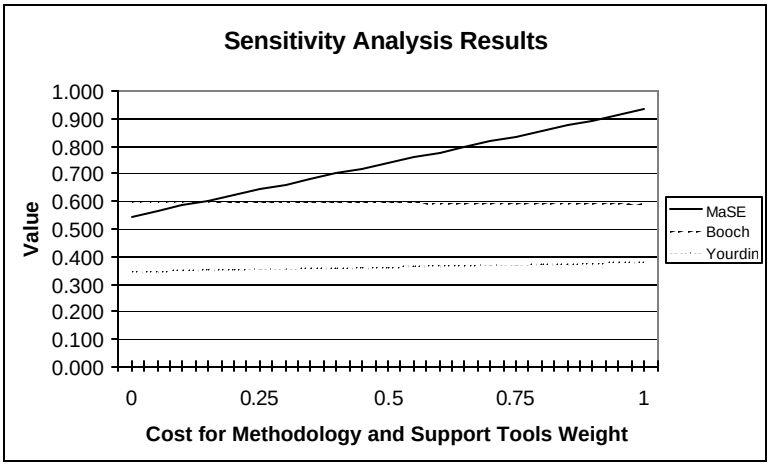

Sensitivity Analysis Results
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Agility & Robustness Weight
MaSE
Booch
Yourdin

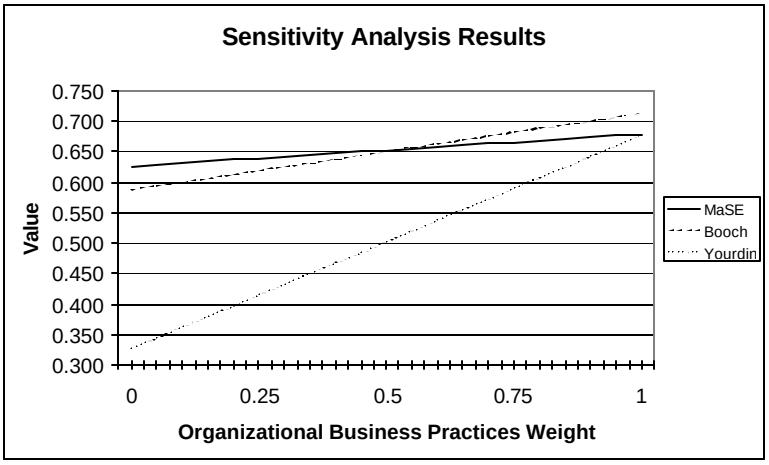

Sensitivity Analysis Results
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Dynamic Structure and Scalability Weight
MaSE
Booch
Yourdin

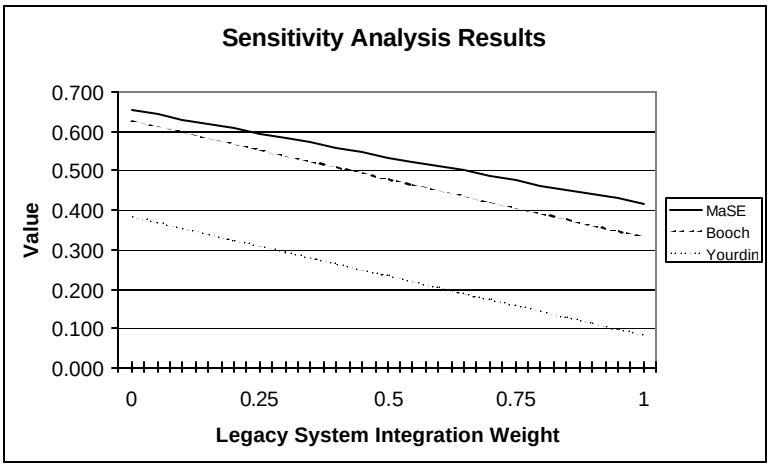

Sensitivity Analysis Results
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Interaction Weight
MaSE
Booch
Yourdin

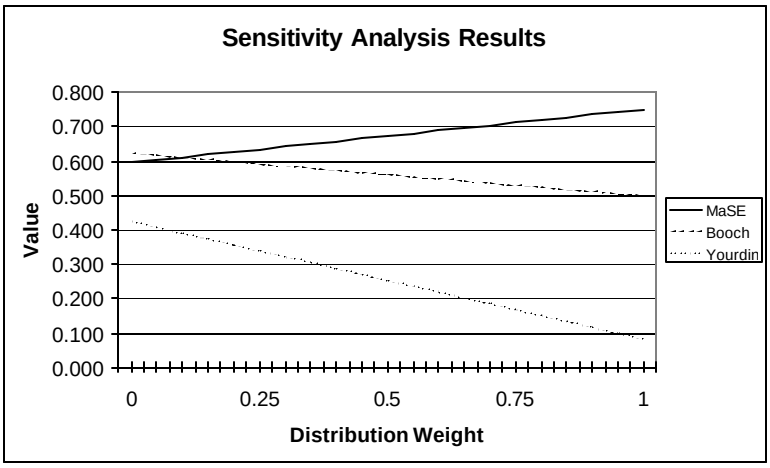

Sensitivity Analysis Results
1.200
1.000
0.800
0.600
0.400
0.200
0.000
Value
0
0.25
0.5
0.75
1
Methodology Maturity Weight
MaSE
Booch
Yourdin

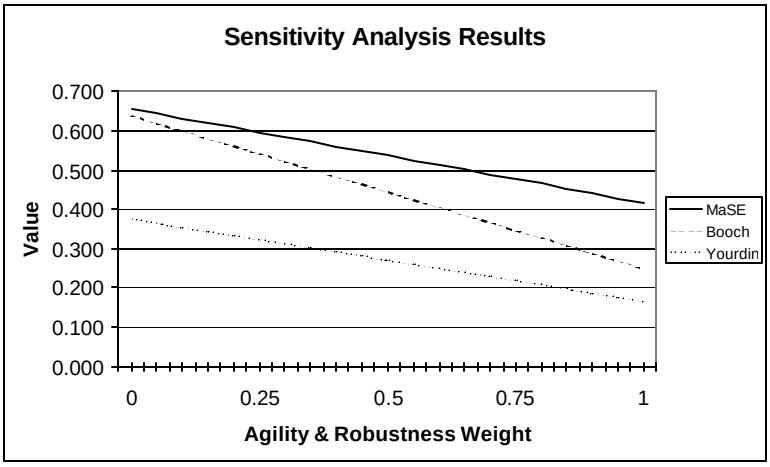
C.4 Case Study 4

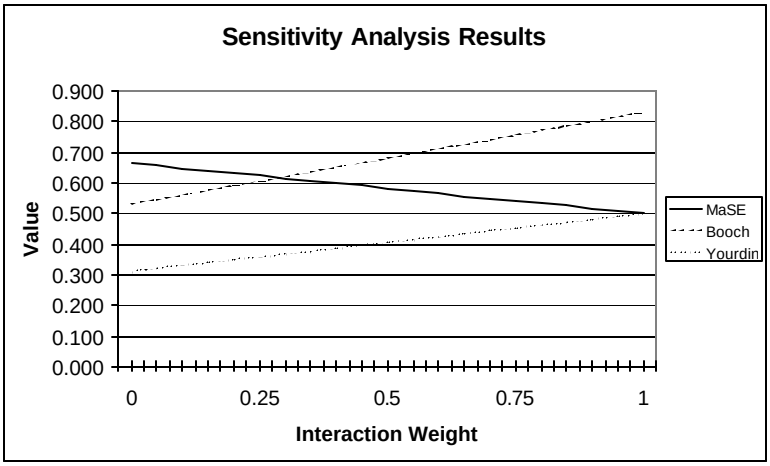


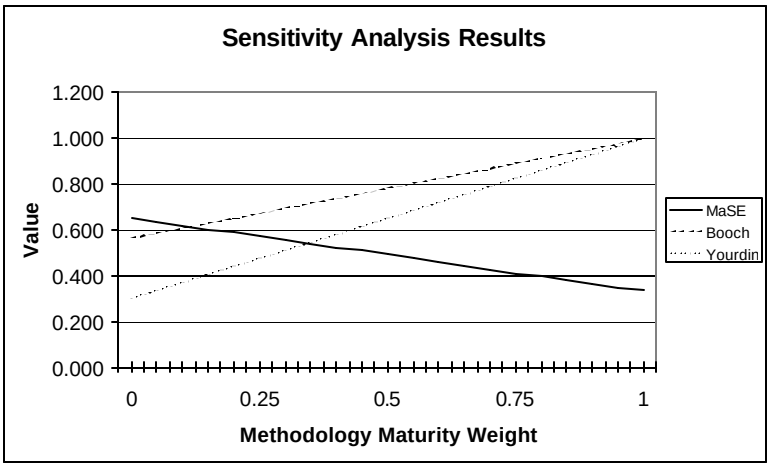





Sensitivity Analysis Results
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Distribution Weight
MaSE
Booch
Yourdin


Sensitivity Analysis Results
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Agility & Robustness Weight
MaSE
Booch
Yourdin


Sensitivity Analysis Results
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
Value
0
0.25
0.5
0.75
1
Interaction Weight
MaSE
Booch
Yourdin


Sensitivity Analysis Results
1.200
1.000
0.800
0.600
0.400
0.200
0.000
Value
0
0.25
0.5
0.75
1
Methodology Maturity Weight
MaSE
Booch
Yourdin

Appendix D. Software Engineering Methodology Decision Analysis Tool (SEM-DAT) and Data Analyzer User's Manual

D.1 Introduction

The Software Engineering Methodology Decision Analysis Tool (SEM-DAT) is a software project developed to assist decision makers utilizing the decision-making framework established in this thesis. The application ensures that the process is followed appropriately. In addition, it supports the decision maker by automating a number of the calculations that would otherwise have to be computed by hand. The Data Analyzer tool is an auxiliary tool that provides graphical representations of the decision, and sensitivity analysis support.

The SEM-DAT application is a Java program. The main focus of the program is to ensure the collection of data to on which the methodology selection decision is based. The Data Analyzer tool is an analysis application developed in Microsoft Excel. This software package was chosen for its ability to quickly represent data in charts.

D.2 Installation

SEM-DAT and Data Analyzer are packaged in a zip file called DATv1.zip. When expanded the following directories will be created:

- ?? *DATSoftwareTool*\
- ?? *DATSoftwareToolAnalyzer*\
- ?? *DATSoftwareToolInterface*\

The working directory selected will also have two batch files used to facilitate use of the software. Executing *DAT-start.bat* starts the software.

System and software requirements for the using the system are as follows (the options underlined indicate the software used for development):

- ?? Hardware – Intel-based Pentium processor
- ?? Operating System – Windows NT 4.0, Windows 9x
- ?? Java Version – 1.2 or higher
- ?? Microsoft Excel Version – 95, 98 or 2000

D.3 Features

SEM-DAT is a five-step process for calculating the multiobjective functions of software methodology alternatives. It is currently structure to compare two alternatives. Each of the five steps is discussed below. In order to move from one step to the next step, buttons are provided at the bottom right corner of the window. The buttons do perform some testing of the data in each step, so the next step will not be activated unless the correct data has been entered into the system.

D.3.1 STEP 1

Step 1 is the first step in the "Weighting" phase of the decision analysis process. A list of the evaluation considerations that the tool supports is provided in the possible criteria field. For the specific software requirements problem, the relevant considerations are selected by highlighting the considerations individually, and using the ">>>" button. If a consideration is placed in the selected box by error, it can be returned to the possible criteria field by highlighting it and using the "<<<" button. At least one consideration must be selected. After the relevant considerations have been selected, go to step 2.

D.3.2 STEP 2

The next step in the weighting process is ranking the evaluation considerations based on their importance to the decision. Selecting a consideration and using the "move up and move down" orders the list. The considerations should be ordered so the most important considerations are at the top of the list. When the considerations have been ordered, go to step 3.

D.3.3 STEP 3

The final step in the weighting process is assigning a weight for each evaluation consideration relative to the next least important evaluation consideration. The next to the least important and the least important considerations are compared first. Values that should be entered are real numbers greater than or equal to one. For considerations that are "equal" should be given a rating of 1. These ratings are normalized at the end of this step.

D.3.4 STEP 4

The next step of SEM-DAT represents the “Rating” phase of the decision-making process. For each of the evaluation considerations selected in Step 1 a set of focus points are presented in a unique tab. The focus points should be rated on a scale of 0 to 4 where 0 indicates "No Support" for the question, 4 indicates "Full Support", and the values in between provide for partial support. The first column of input fields represents alternative 1 and the second column represents alternative 2. If a focus point is not to be considered in the decision process, the field should be left blank. Before moving on to Step 5, each evaluation consideration should be rated.

D.3.5 STEP 5

Step 5 provides an opportunity to review the choices made up to this point. Each of the considerations is listed in the table along with the normalized weights, which have been calculated from the relative weights assigned in Step 3, and the ratings of each alternative. You can continue and get the results by pressing the "Ratings" button, or go back to any of the previous steps and modify your responses. If modifications are necessary, the data collected in step can be changed by selecting the tab for that step. After making the change, the button in the bottom right corner should be used to move back through to Step 5. If additional considerations are added in Step 1, then all the steps after must be reaccomplished. The response to the focus points will be maintained for any consideration previously evaluated.

D.3.6 RATING

The final multiobjective fitness scores are reported on this screen. The decision-maker can choose to go back to any previous stage and make changes, choose to use the Data Analyzer, or exit the program. The Data Analyzer is invoked by using the "Save" button. Microsoft Excel should start at this time. In the event that it does not start, it can be manually started by executing Excel and opening the file *datv1.xls*, which can be found in the directory *\DATSoftwareToolAnalyzer*.

D.3.7 Data Analyzer

After invoking the Data Analyzer, a dialogue will open asking if macros should be enabled. Enabling macros will allow for refreshing the input data. When the file completes, select the worksheet called "Input" and press "Ctrl-r". This will invoke the macro for refreshing the data. The input data is in a file called *output.dat* in the working directory. After the data is updated, the graphical representations and sensitivity analyses can be viewed in the "Data" and "Charts" worksheets, respectively.

D.4 Demonstration

This section demonstrates the execution of SEM-DAT and the Data Analyzer by applying the first case study from Section 4.3.2. Assuming the system is installed as described in Section D.2, after executing the batch file, *DAT-start.bat*, the application will begin with the screen shown in Figure 58.

Figure 58: SEM-DAT Welcome Screen

The Welcome Screen provides the user with information about what the system does, as well as information describing the process the user is to follow. To begin the analysis process, press the button "Step 1" at the bottom right corner of the screen. The tab is shown in Figure 59. The user selects the evaluation considerations relevant to the problem. For this problem, the user selects the following evaluation considerations: *Cost of Methodology and Support Tools*, *Organizational Business Practices*, *Methodology Maturity*, *Distribution*, *Environment*, *Agility and Robustness*, *Dynamic Structure and Scalability*, and *Interaction.*

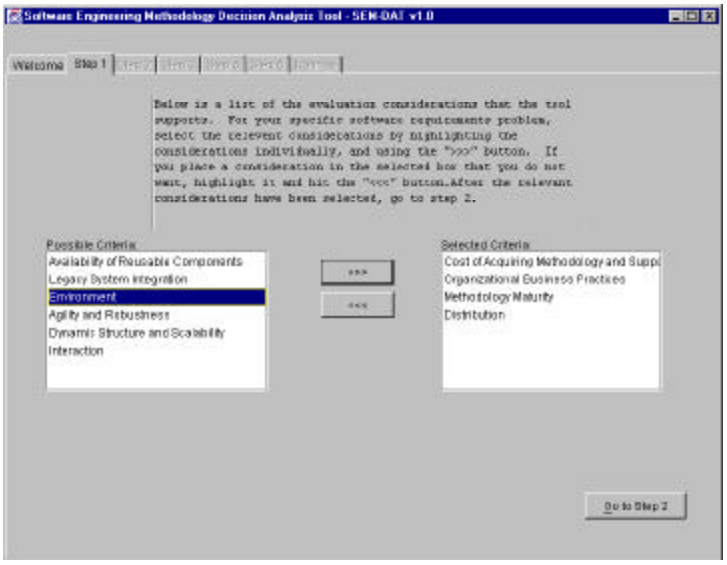


Figure 59: SEM-DAT Step 1 Screen

After the considerations are selected, the user hits the "Step 2" button. The next tab, shown in Figure 60, becomes active. This tab allows the user to order the evaluation considerations based on their importance to the decision. The order of importance, selected for this case, is the following: *Cost of Methodology and Support Tools*, *Distribution*, *Environment*, *Interaction*, *Agility and Robustness*, *Dynamic Structure and Scalability*, *Methodology Maturity*, and *Organizational Business Practices.*

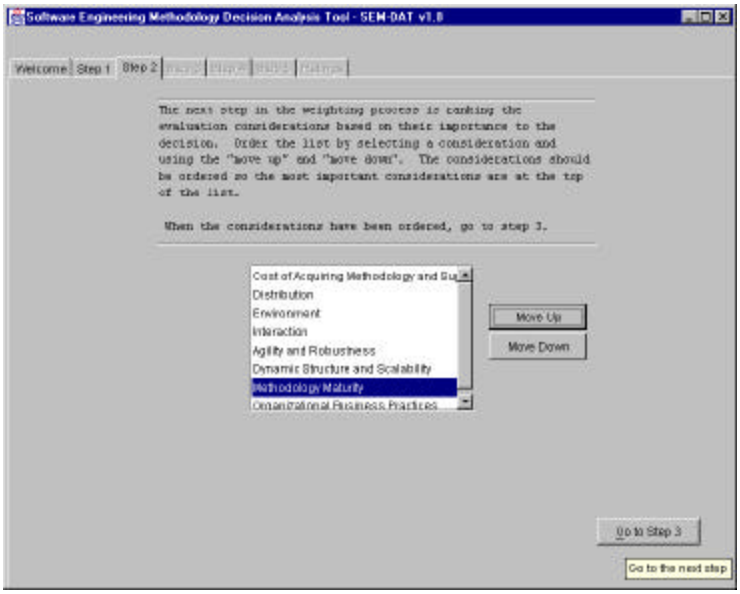


Figure 60: SEM-DAT Step 2 Screen

The next tab is activated after the user hits the "Step 3" button. This tab allows for the input of relative weights. The relative weights have been captured in Figure 36. Upon the completion of this step, the user hits the "Step 4" button.

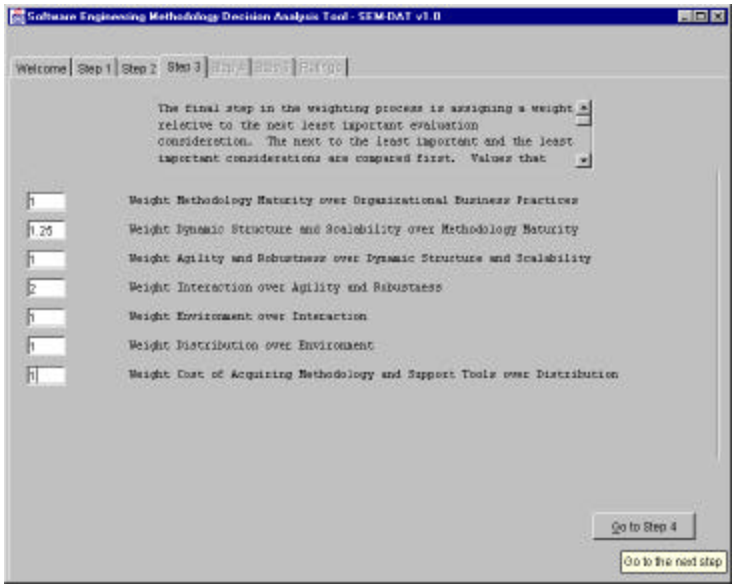


Figure 61: SEM-DAT Step 3 Screen

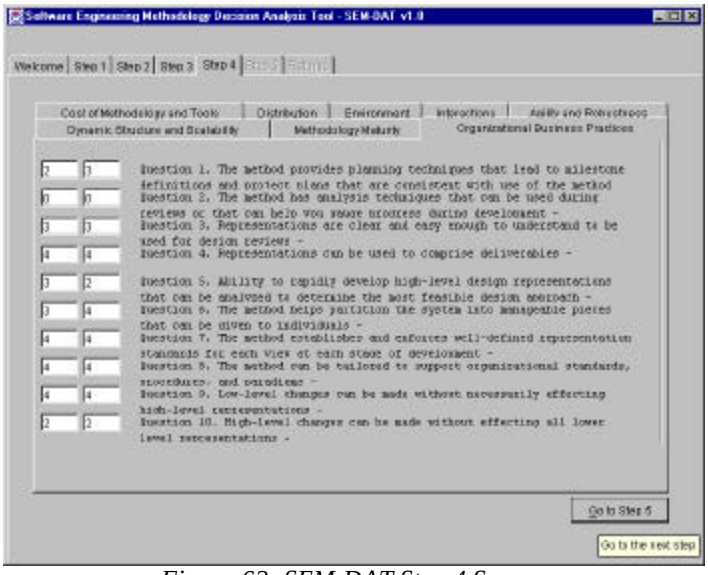


Figure 62: SEM-DAT Step 4 Screen

The "Step 4" tab, shown in Figure 62, list the focus points for the evaluation considerations selected in step 1. The responses to each set of questions can be found in the appropriate considerations in Figure 25 to Figure 34. After entering the data, the user hits the "Step 4" button. The tab, shown in Figure 63, provides the user with an opportunity to review the data entered.

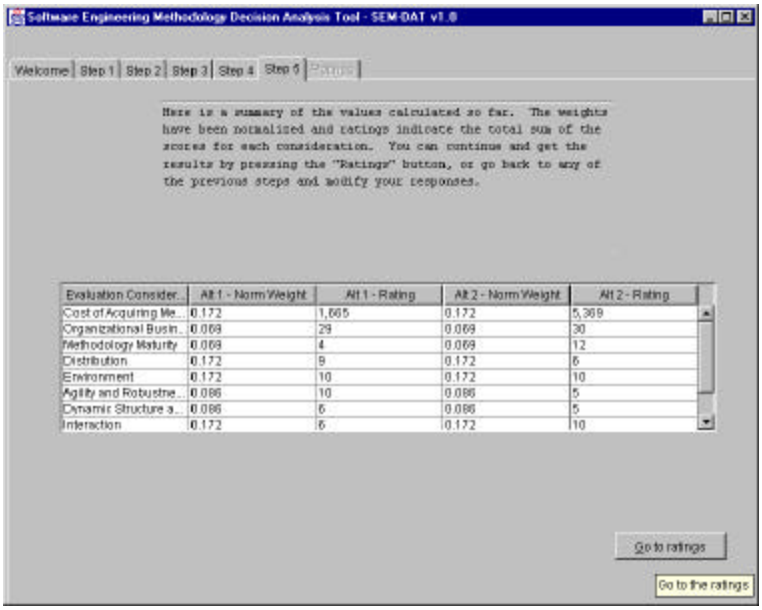


Figure 63: SEM-DAT Step 5 Screen

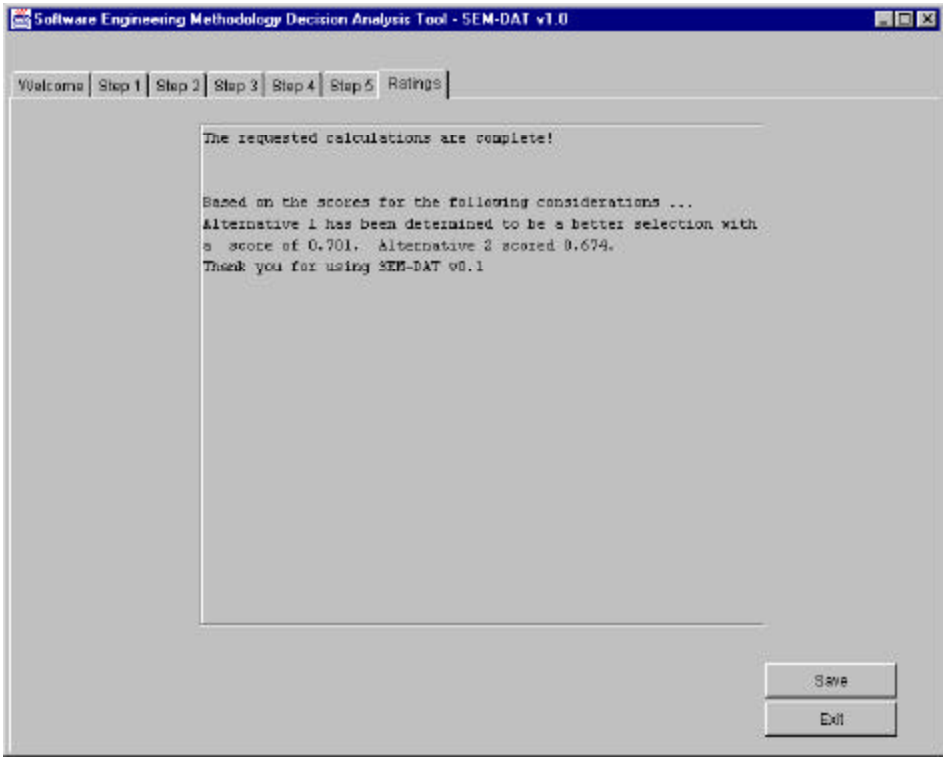


Figure 64: SEM-DAT Ratings Screen

After reviewing the data, the user hits the "Ratings" button. The result of the multiobjective analysis is presented. The user can now accept this result as is, or return to any step in the process and make changes to the input. A third option is to use the data analyzer. The data analyzer is launched by hitting the "Save" button. The data analyzer provides a series of sensitivity analyses of the user's consideration. When the data analyzer opens, the first worksheet is the data input sheet. It is shown in Figure 65. The data is updated by using a macro; it is invoked by hitting "Ctrl-r".

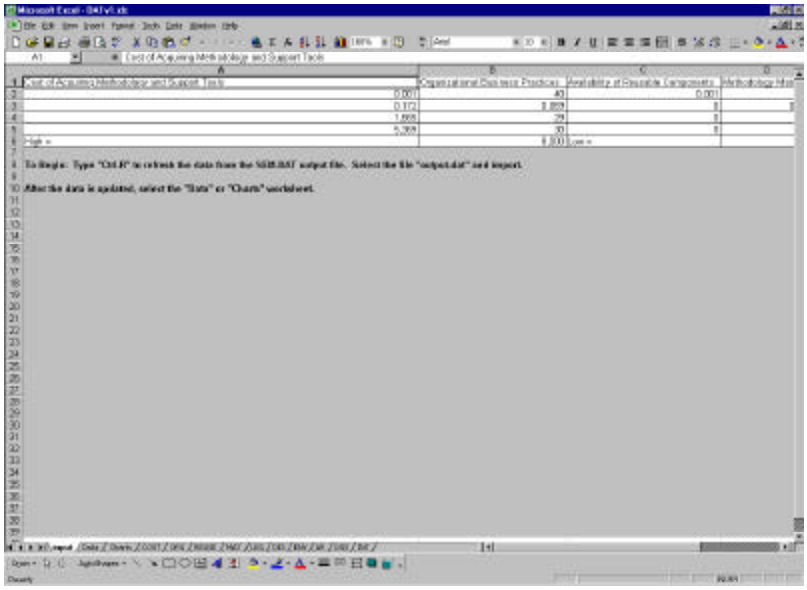


Figure 65: Data Analyzer Data Input Screen

After updating the data, the data analyzer presents the user with graphical representations of the choices the user made, such as the percentage of weight for each consideration and the breakdown of the score each alternative received as seen in Figure 66. The sensitivity analyses are summarized in the "Charts" worksheet, see Figure 67. The sensitivity analysis for each consideration can also be found in the worksheet named following the conventions in Figure 24.

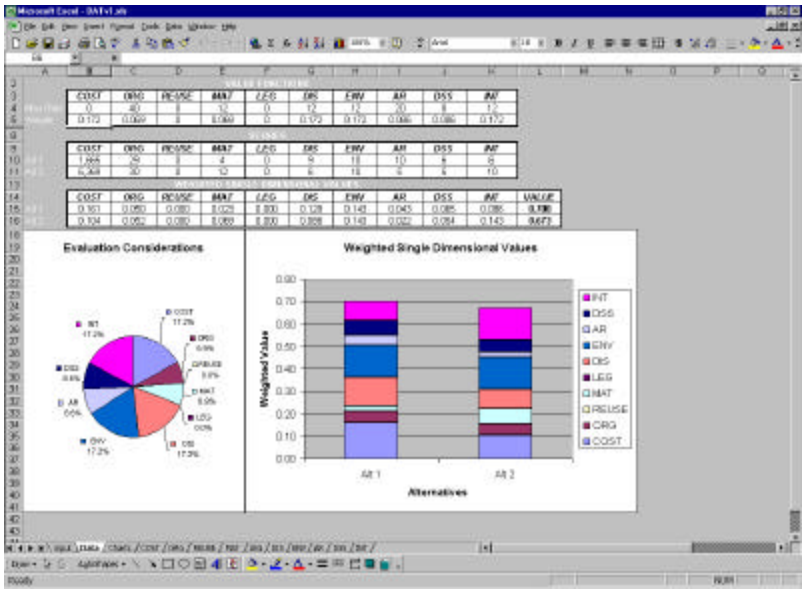


Figure 66: Data Analyzer Data Evaluation Screen

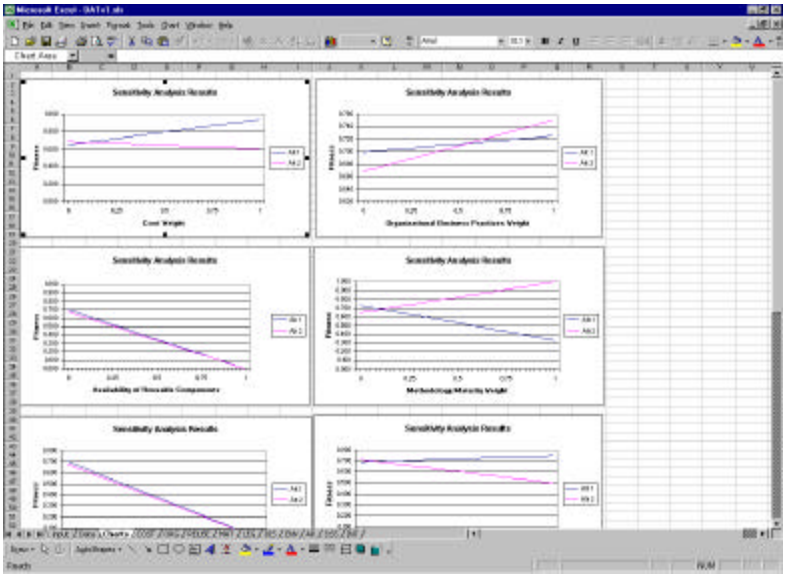
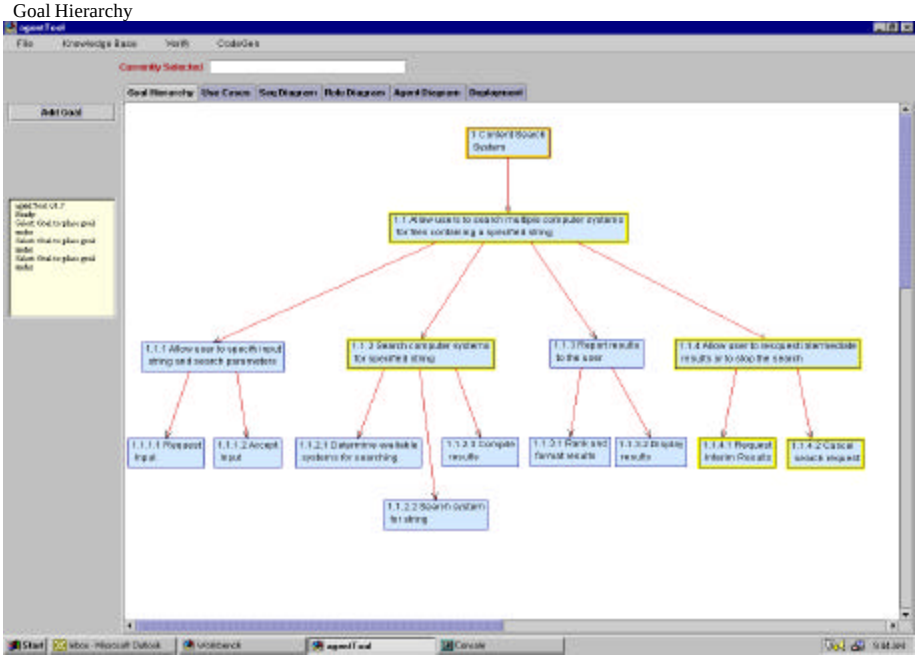


Figure 67: Data Analyzer Charts Screen

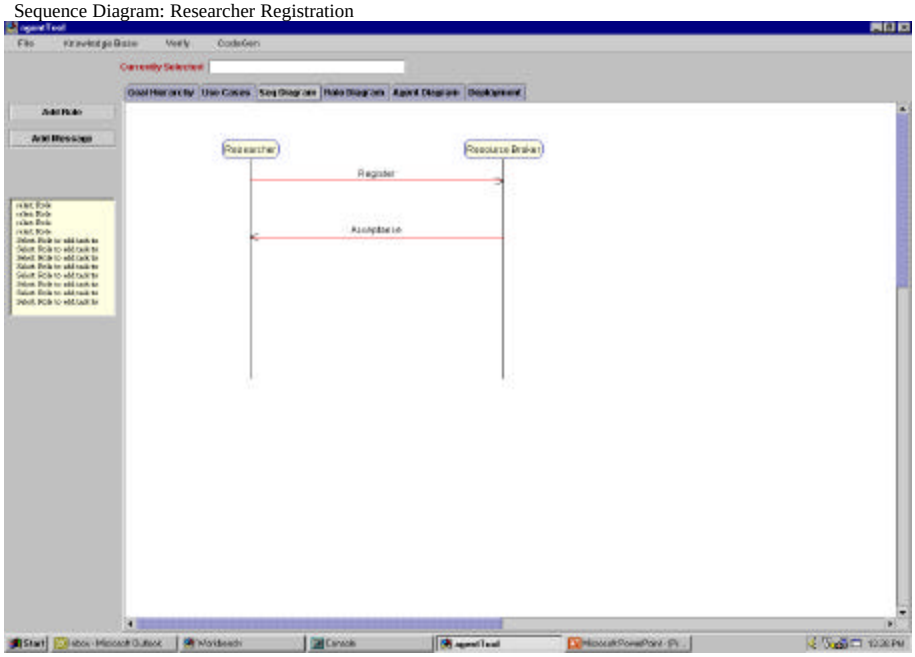
Appendix E. Software Development Questionnaire

E.1 Content Search System Analysis and Design Models – MaSE

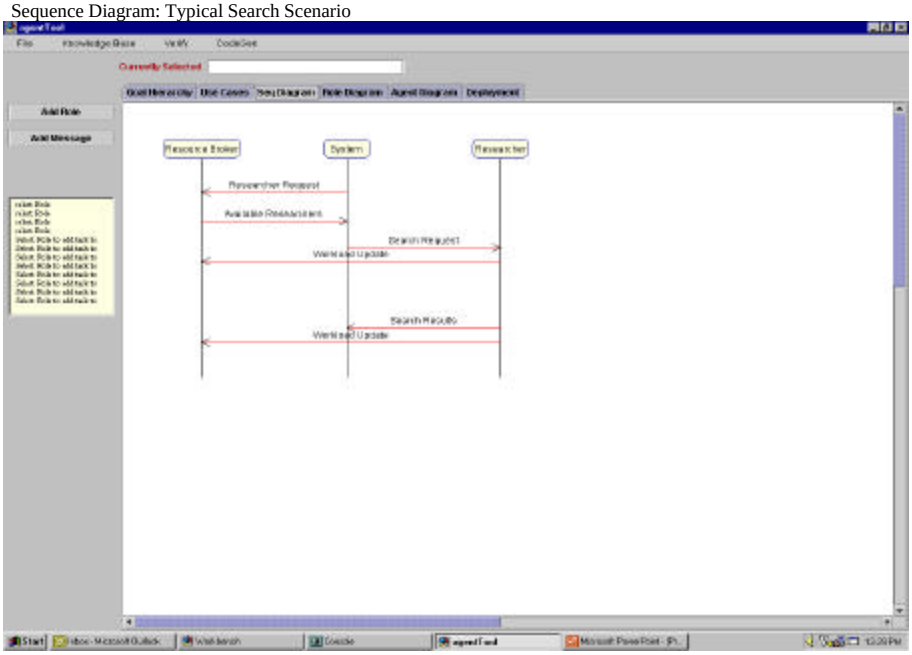
Goal Hierarchy



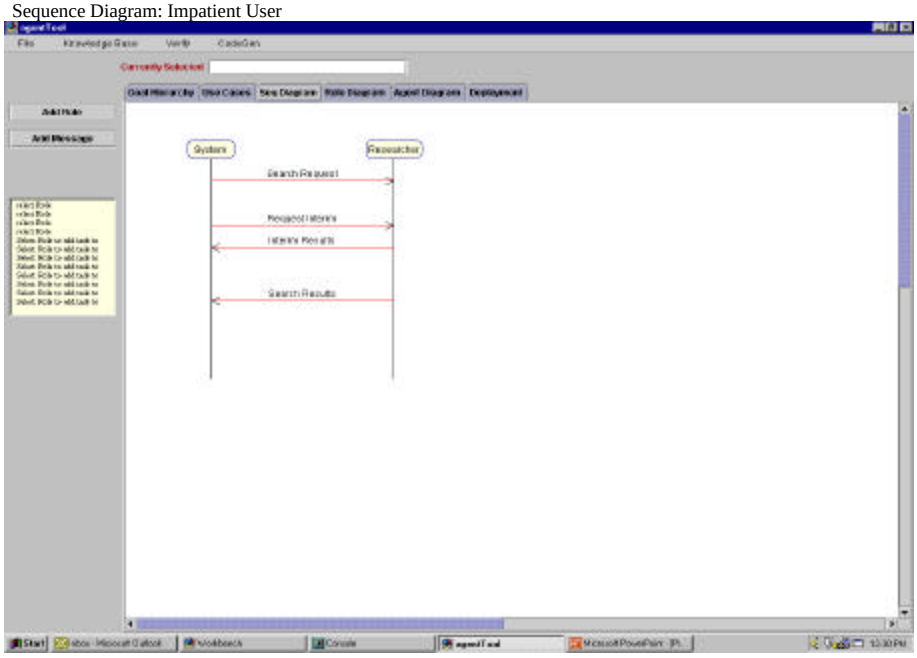
Sequence Diagram: Researcher Registration



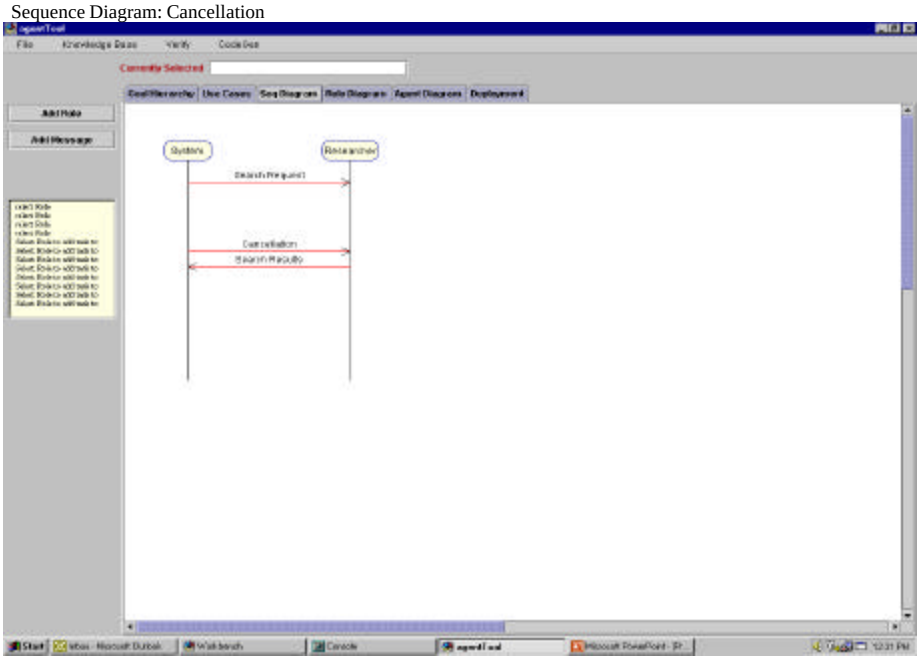
Sequence Diagram: Typical Search Scenario



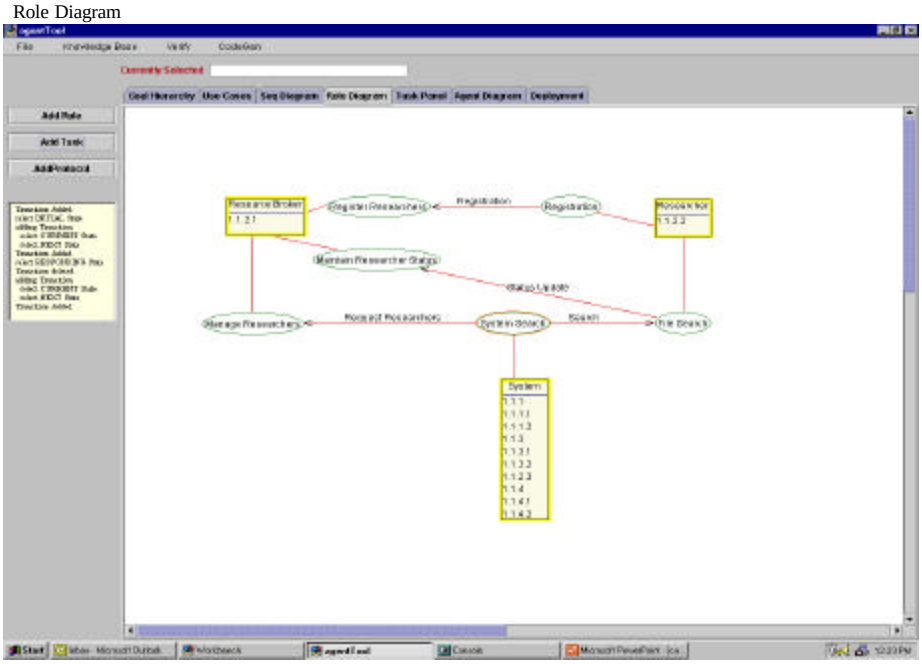
Sequence Diagram: Impatient User



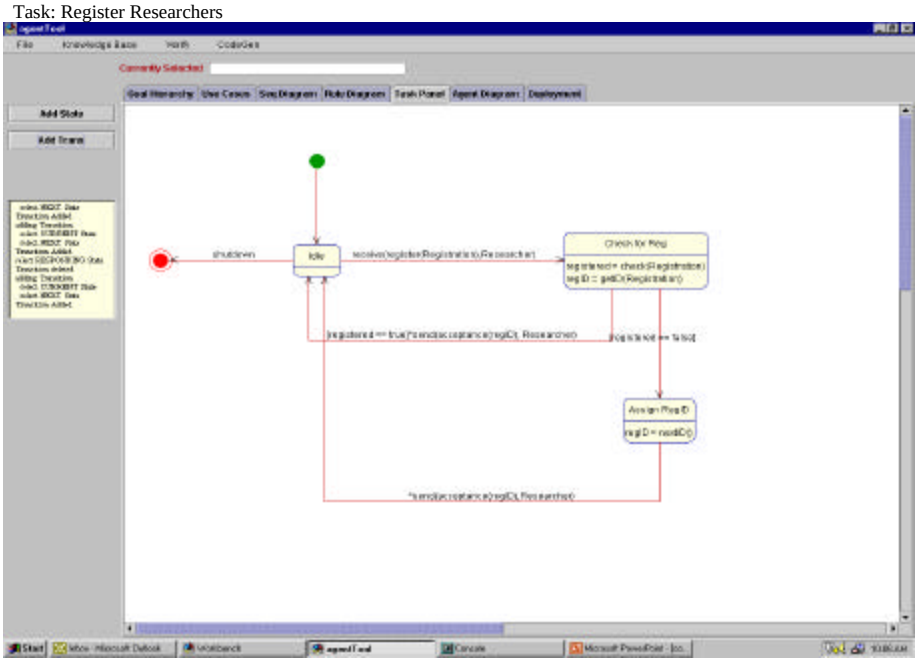
Sequence Diagram: Cancellation



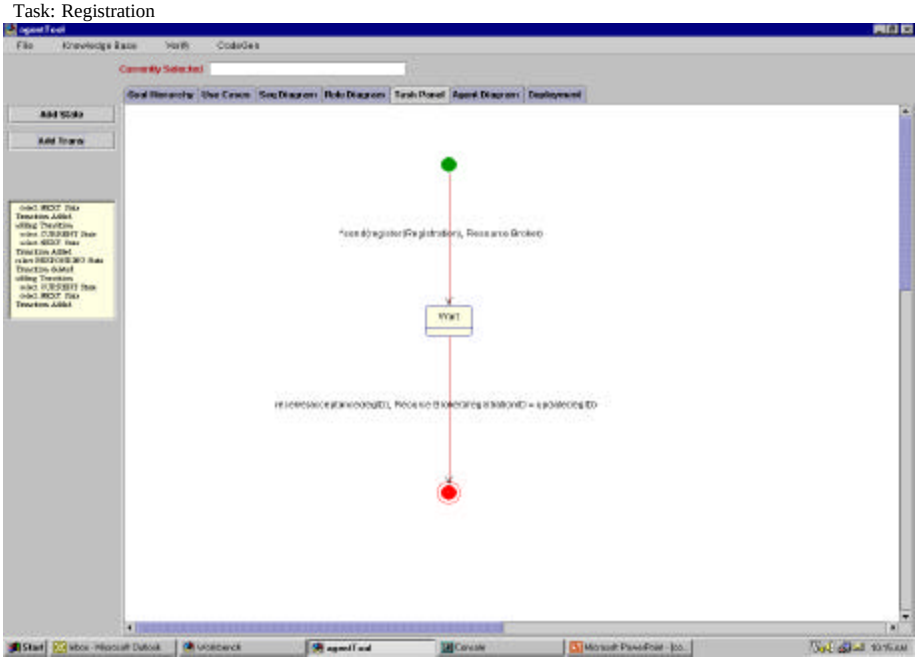
Role Diagram



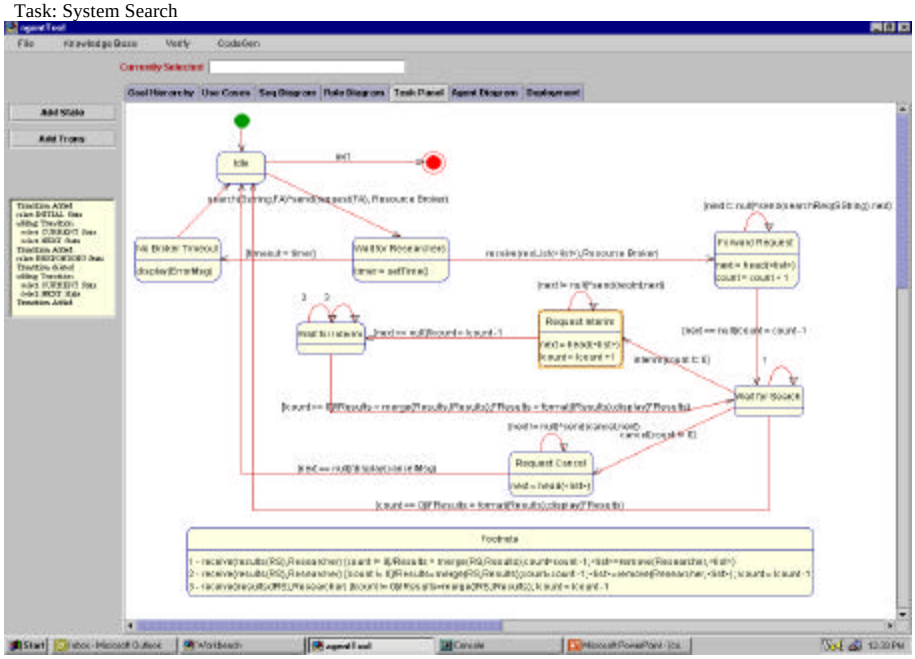
Task: Register Researchers



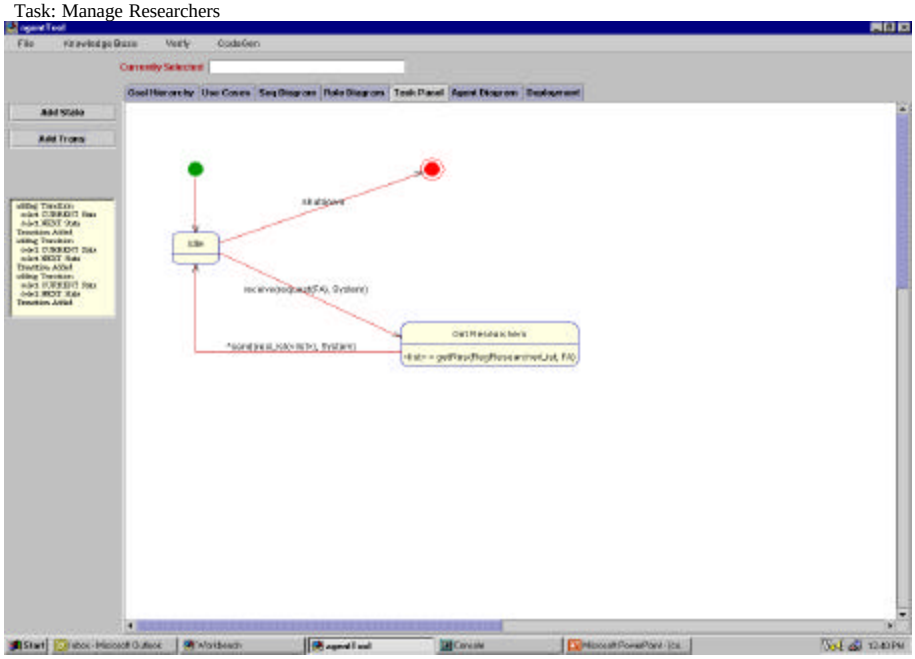
Task: Registration



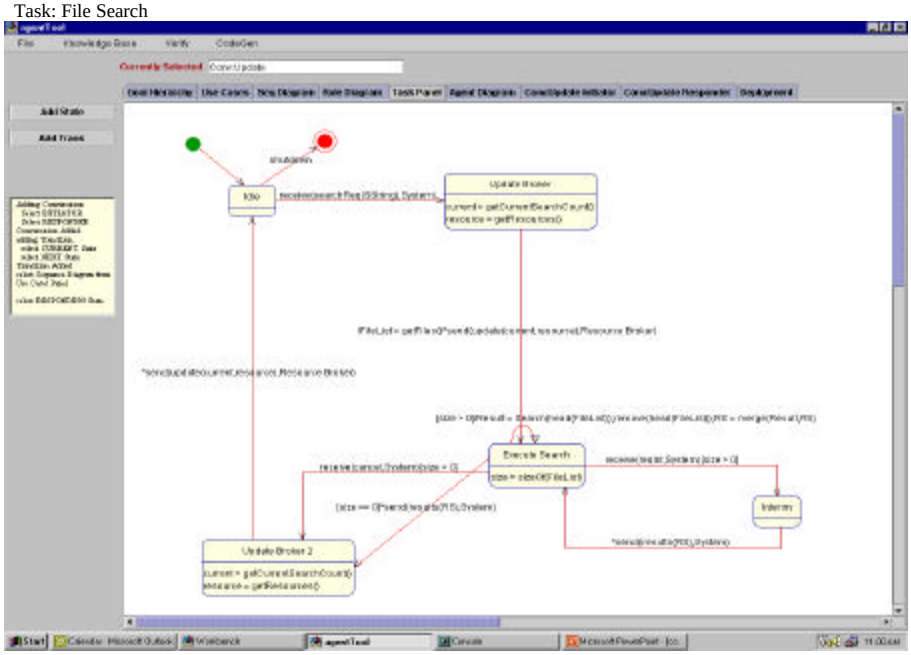
Task: System Search



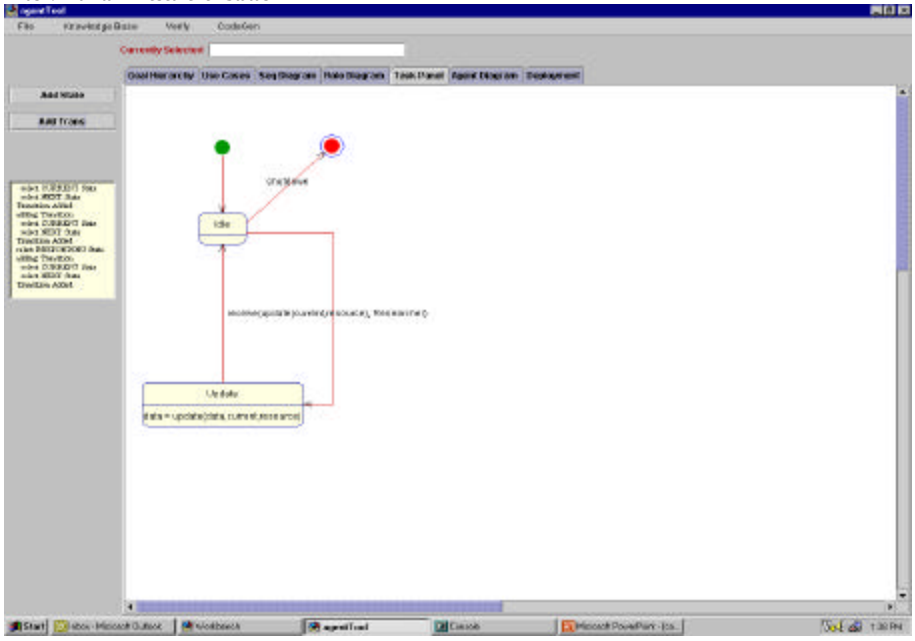
Task: Manage Researchers



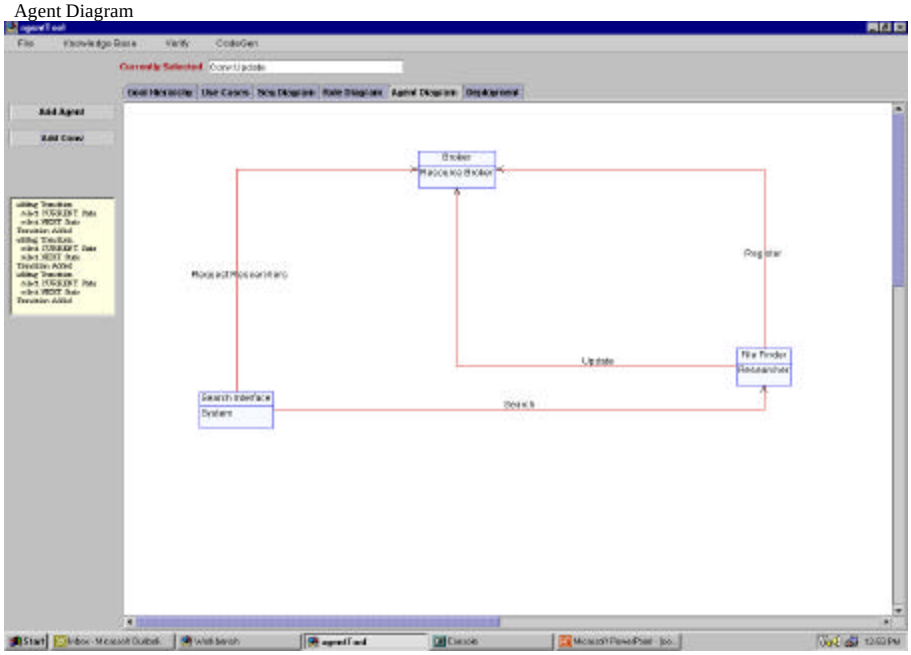
Task: File Search



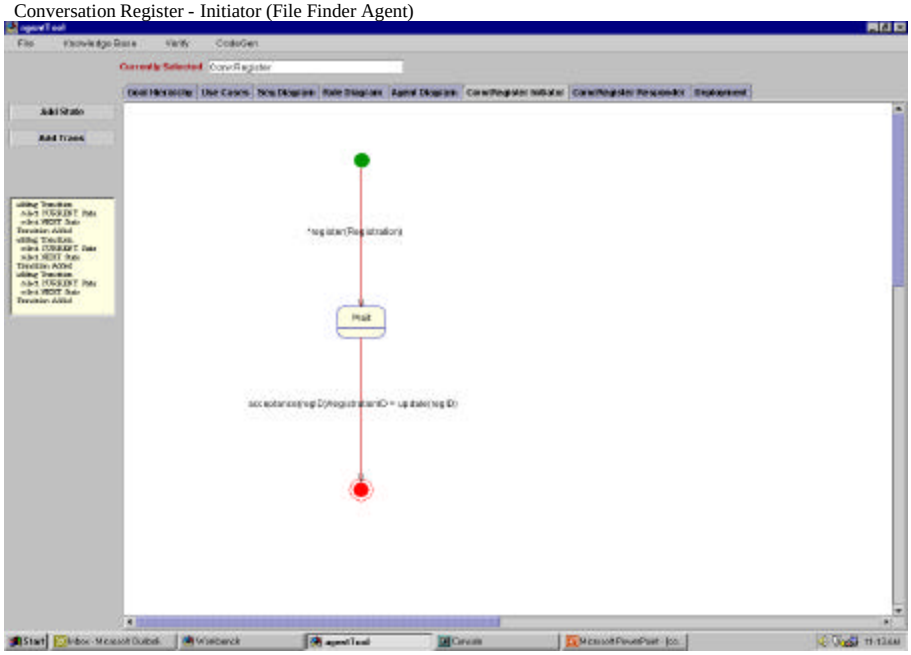
Task: Maintain Researcher Status



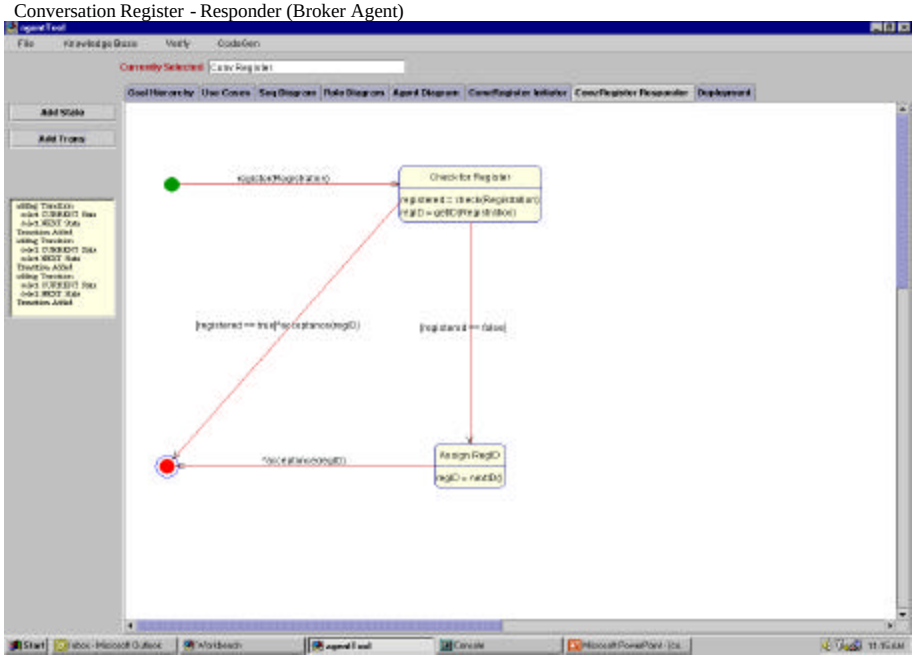
Agent Diagram



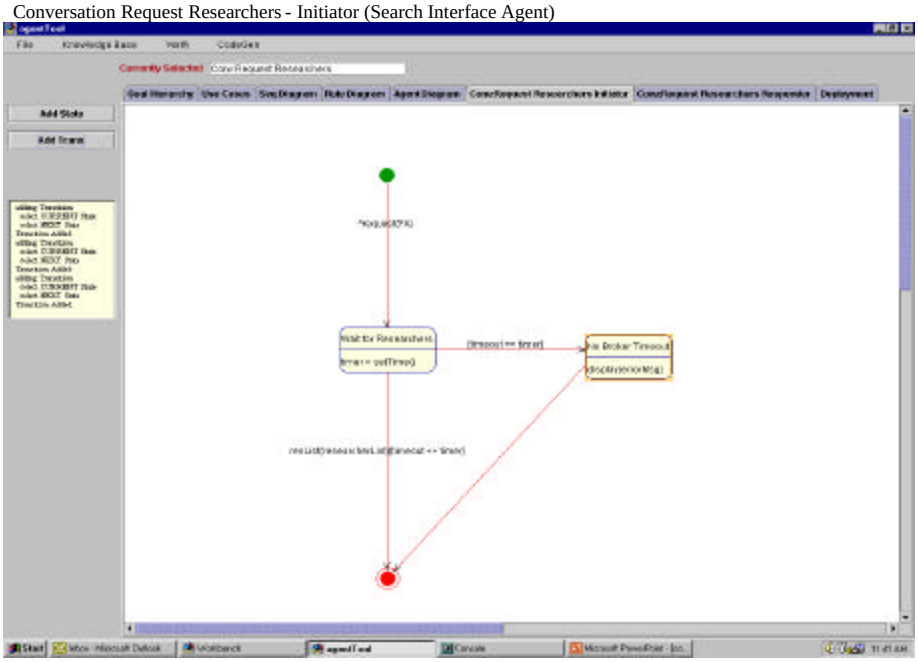
Conversation Register - Initiator (File Finder Agent)



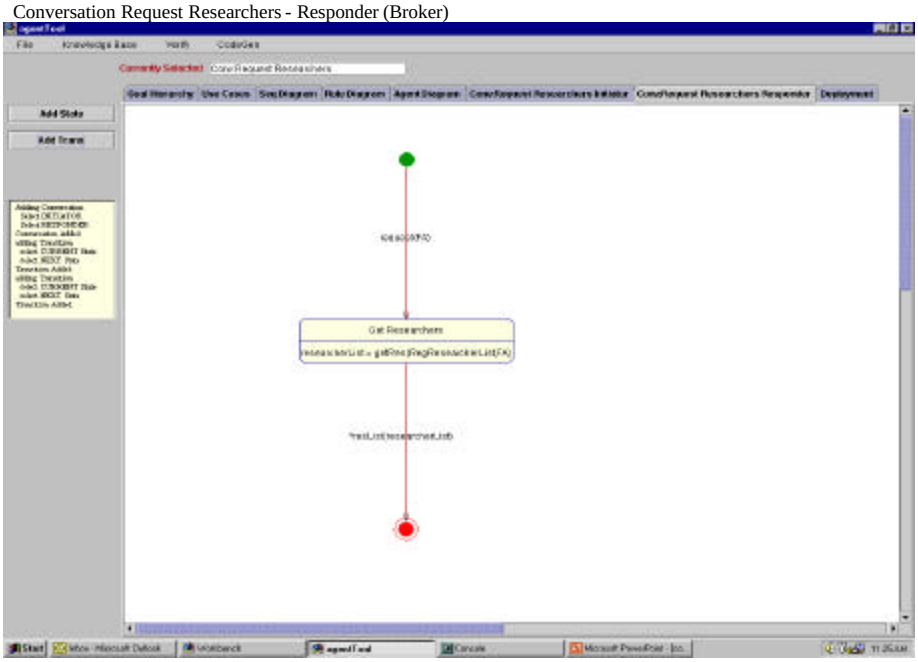
Conversation Register - Responder (Broker Agent)



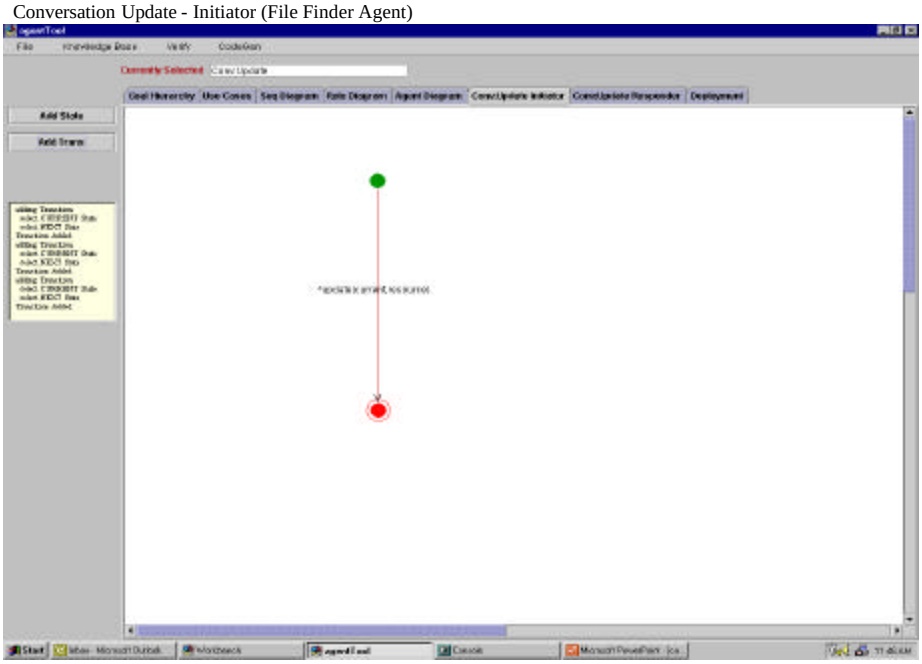
Conversation Request Researchers - Initiator (Search Interface Agent)



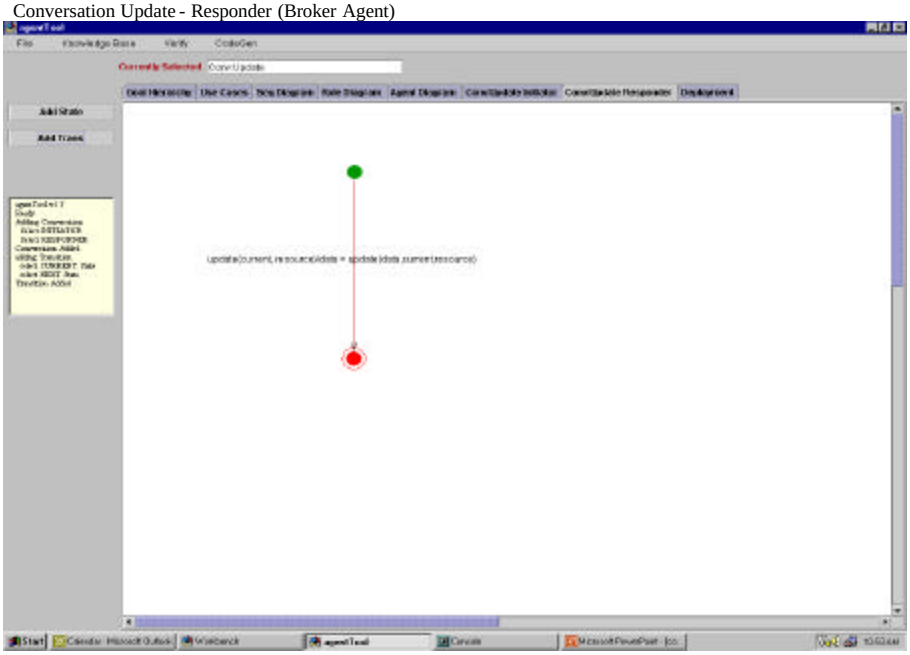
Conversation Request Researchers - Responder (Broker)



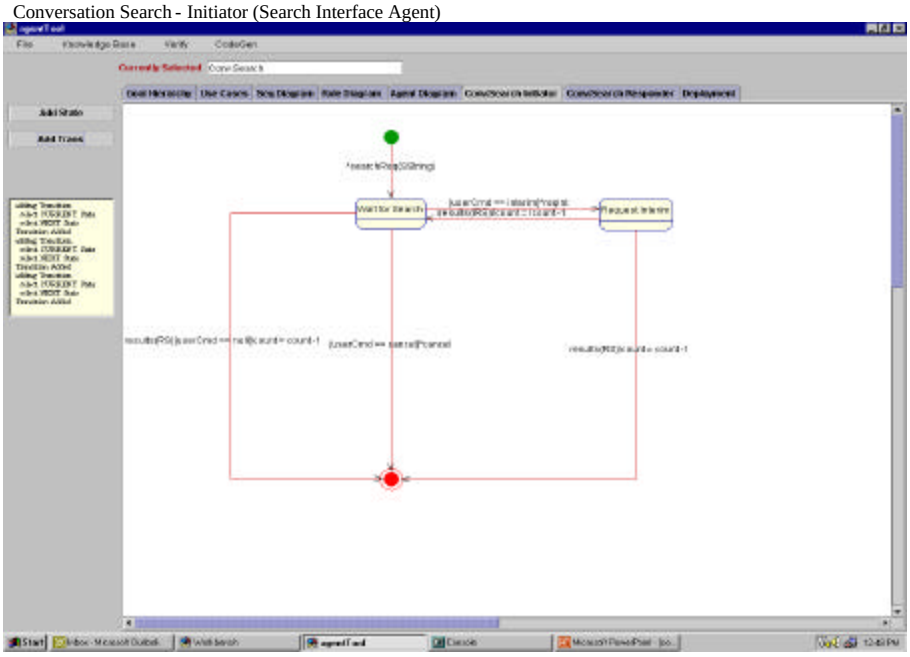
Conversation Update - Initiator (File Finder Agent)



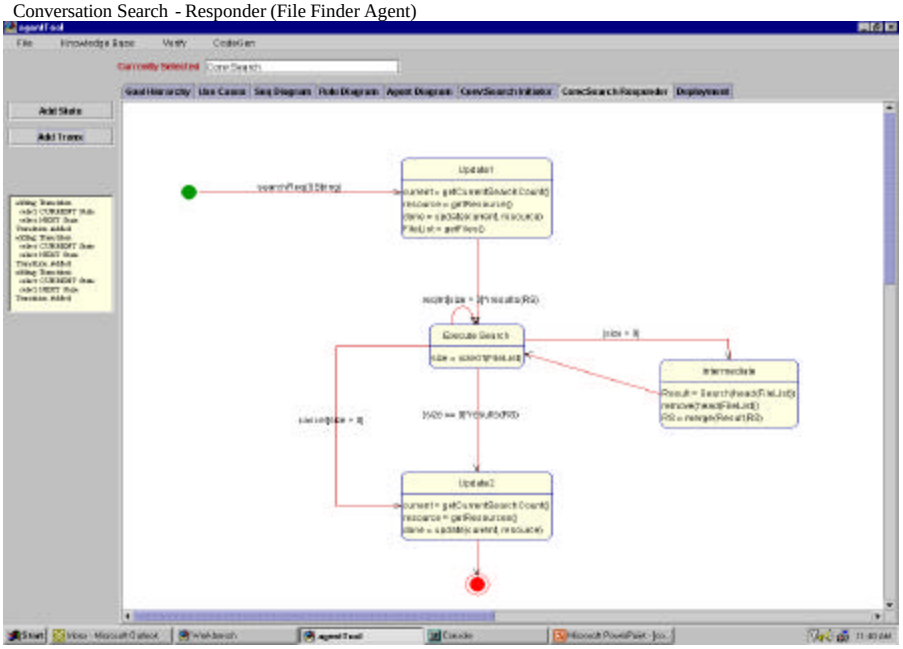
Conversation Update - Responder (Broker Agent)



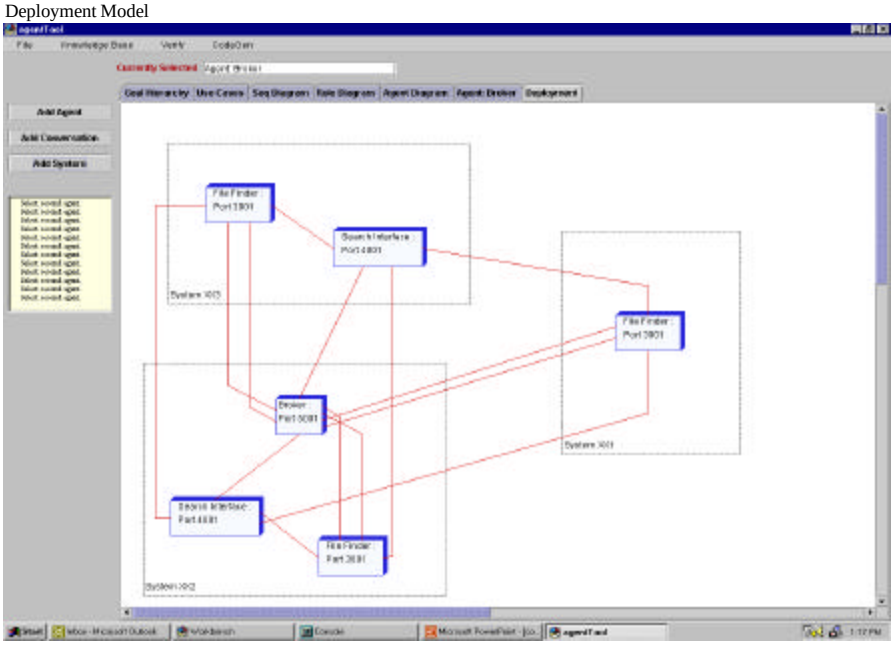
Conversation Search - Initiator (Search Interface Agent)



Conversation Search - Responder (File Finder Agent)



Deployment Model



Data Dictionary

USE CASES
Use Case: Researcher Registration
Description:

When a researcher comes on-line, it should register itself with a resource broker.
Sequence Diagrams:
Researcher Registration

Use Case: Typical Search Scenario
Description:
When a user inputs a search request, the system will request a list of available researchers for the specified functional area from the resource broker. The system will then forward the search request to each of the researchers and await the response from the individual searches. Upon receipt of the request, the researcher notifies the resource broker of its increased workload. After each researcher has completed the search, the results will be ordered and reported to the user. The researcher will notify the resource broker of the job completion.
Sequence Diagrams:
Typical Search Scenario

Use Case: Impatient User
Description:
After submitting a request, the user decides that he would like to see results up to that point. The researchers forward the current results to the system, which in turn reports the interim results.
Sequence Diagrams:
Impatient User

Use Case: Cancellation
Description:
After submitting a request, the user decides that he would like to cancel the search. The researcher ends the search and forwards the results to the system.
Sequence Diagrams:
Cancellation

TASKS
Task: Registration
Description:
After initiation, this task registers the Researcher with the Resource Broker as an available file system researcher.

Role = Researcher
Connected to:
Register Researchers via Registration

Task: File Search
Description:
This task accepts a search request from the system through the search protocol. Then the researcher updates the broker with the number of search requests it has pending. The search request is executed on the file system. After completing the search, the results are returned to the system through the search protocol, and the researcher informs the resource broker that the search has been completed.

Role = Researcher
Connected to:
System Search via Search
Maintain Researcher Status via Status Update

Task: System Search
Description:
This task takes a search request from the system user. First, through the "Request Researchers" protocol, the system requests a list of appropriate file system researchers from the Resource Broker. Then, the search request is forwarded to each of the researchers on the list through the "Search" protocol.

Role = System
Connected to:
Manage Researchers via Request Researchers
File Search via Search

Task: Register Researchers
Description:
The Resource Broker accepts registration through the "Registration" protocol. A registration is an announcement from a researcher announcing its availability and the functional area its respective file system represents.

Role = Resource Broker
Connected to:
Registration via Registration

Task: Maintain Researcher Status
Description:
This task maintains the status of the available researchers. Researchers are not considered available after their workload reaches ten concurrent requests.

Role = Resource Broker
Connected to:
File Search via Status Update

Task: Manage Researchers
Description:
This task receives a request for researcher and returns a list of available researchers that meets the required functional areas.

Role = Resource Broker
Connected to:
System Search via Request Researchers

PROTOCOLS
Protocol: Registration
Description:

A registration is passed from a Researcher to the Resource Broker announcing the availability of service.

Task1 = Registration
Task2 = Register Researchers

Protocol: Request Researchers
Description:
The System requests a list of researchers that are provided by the Resource Broker.

Task1 = System Search
Task2 = Manage Researchers

Protocol: Search
Description:
The Search System forwards a request for search to the file system Researchers and waits for the results.

Task1 = System Search
Task2 = File Search

Protocol: Status Update
Description:
The Researcher informs the Resource Broker of its current search request load.

Task1 = File Search
Task2 = Maintain Researcher Status

SYSTEMS
System: System XX2
Description:
System XX2 is a PC/Windows 2000
IP Address: XXX.X.X.XXX
Host Name: XXX2

Agents:
Broker : Port 5001
Search Interface : Port 4001
File Finder : Port 3001

System: System XX3
Description:
System XX3 is a Spark 10
IP Address: XXX.X.X.XXX
Host Name: XXX3

Agents:
Search Interface : Port 4001
File Finder : Port 3001

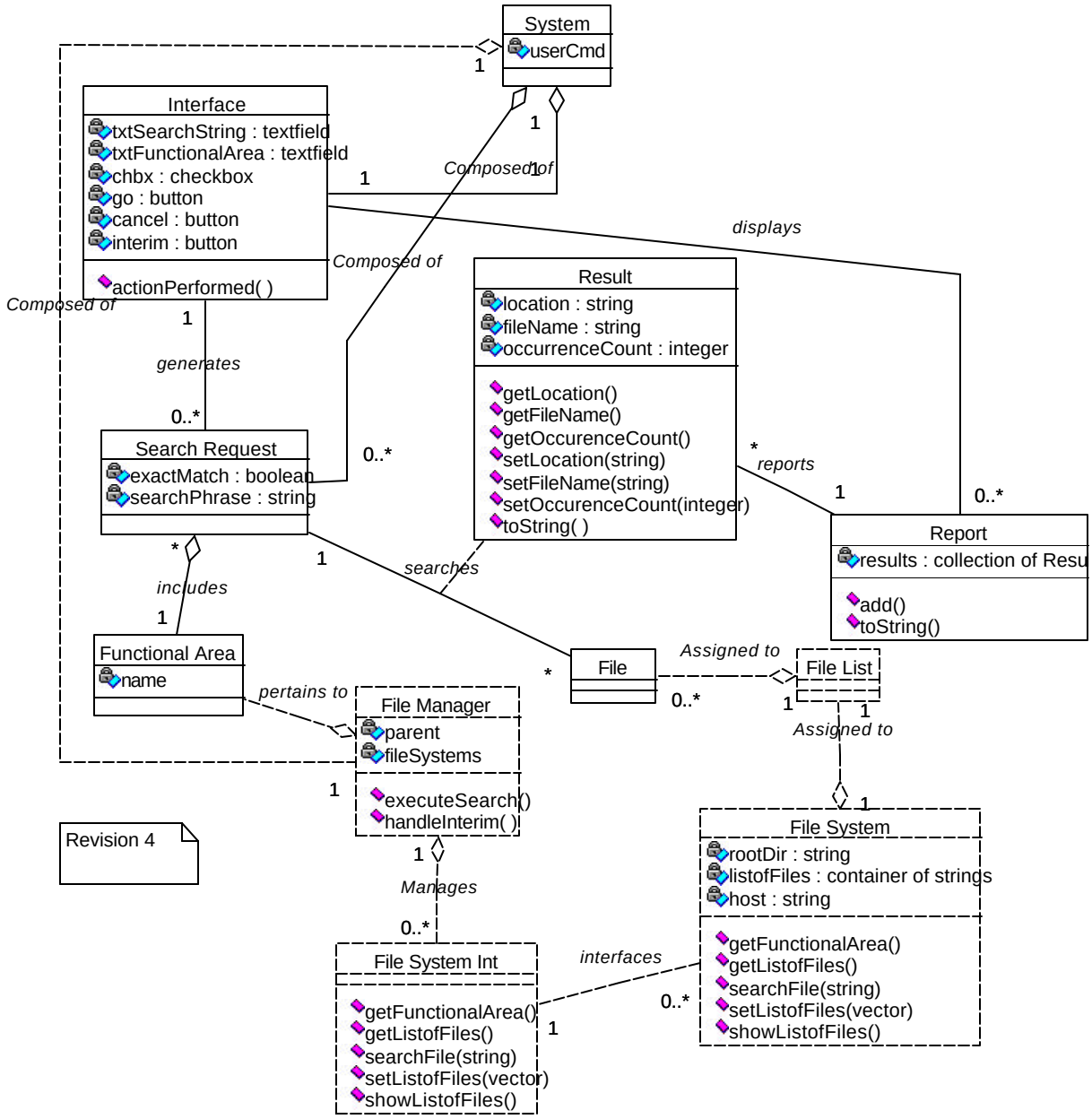
System: System XX1
Description:

System XX1 is a Spark 10
IP Address: XXX.X.X.XXX
Host Name: XXX1

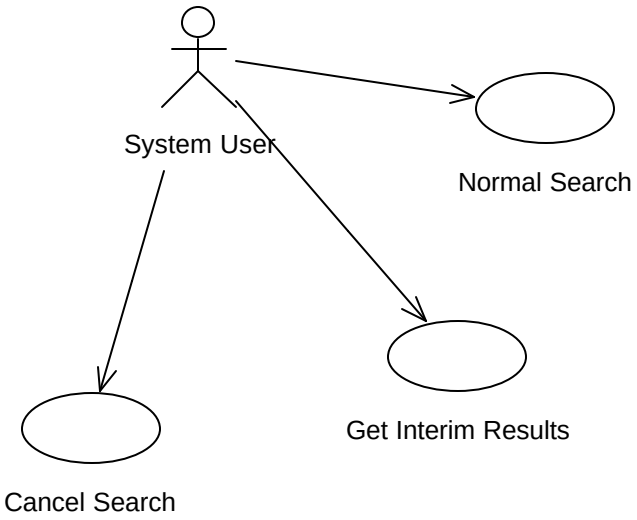
Agents:
File Finder : Port 3001

E.2 Content Search System Analysis and Design Models – Booch

Logical View:

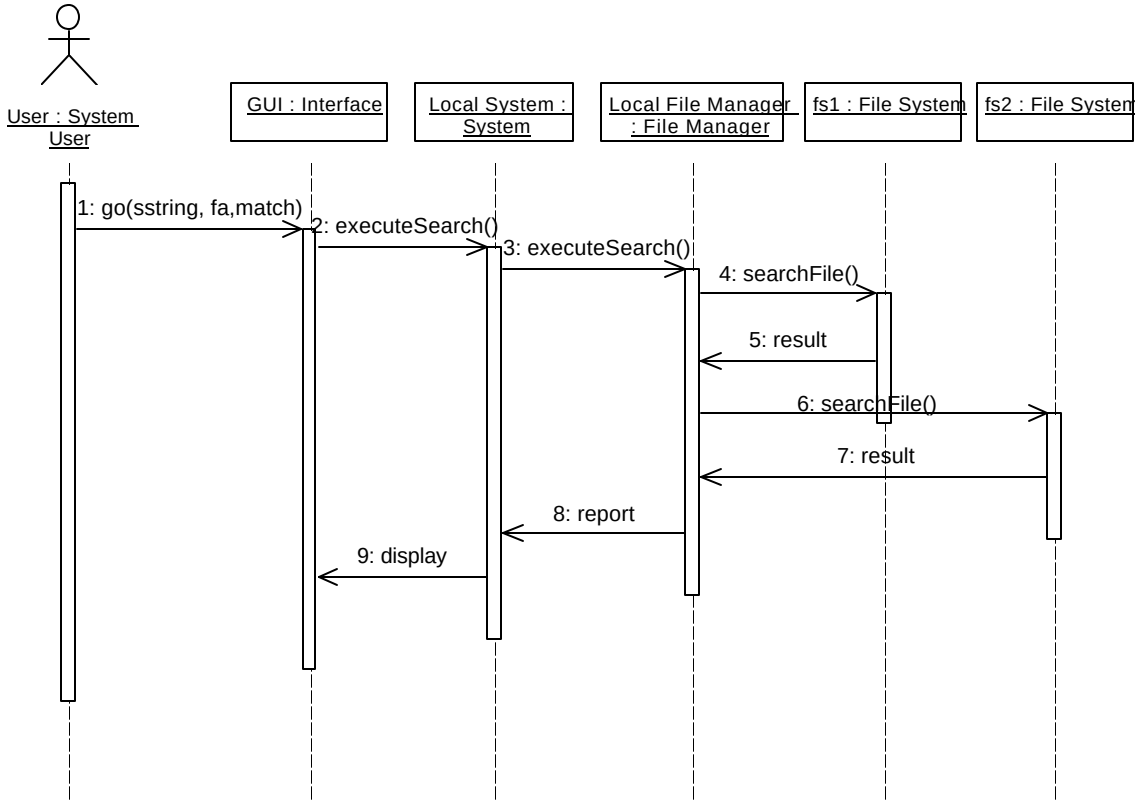


Use Cases:

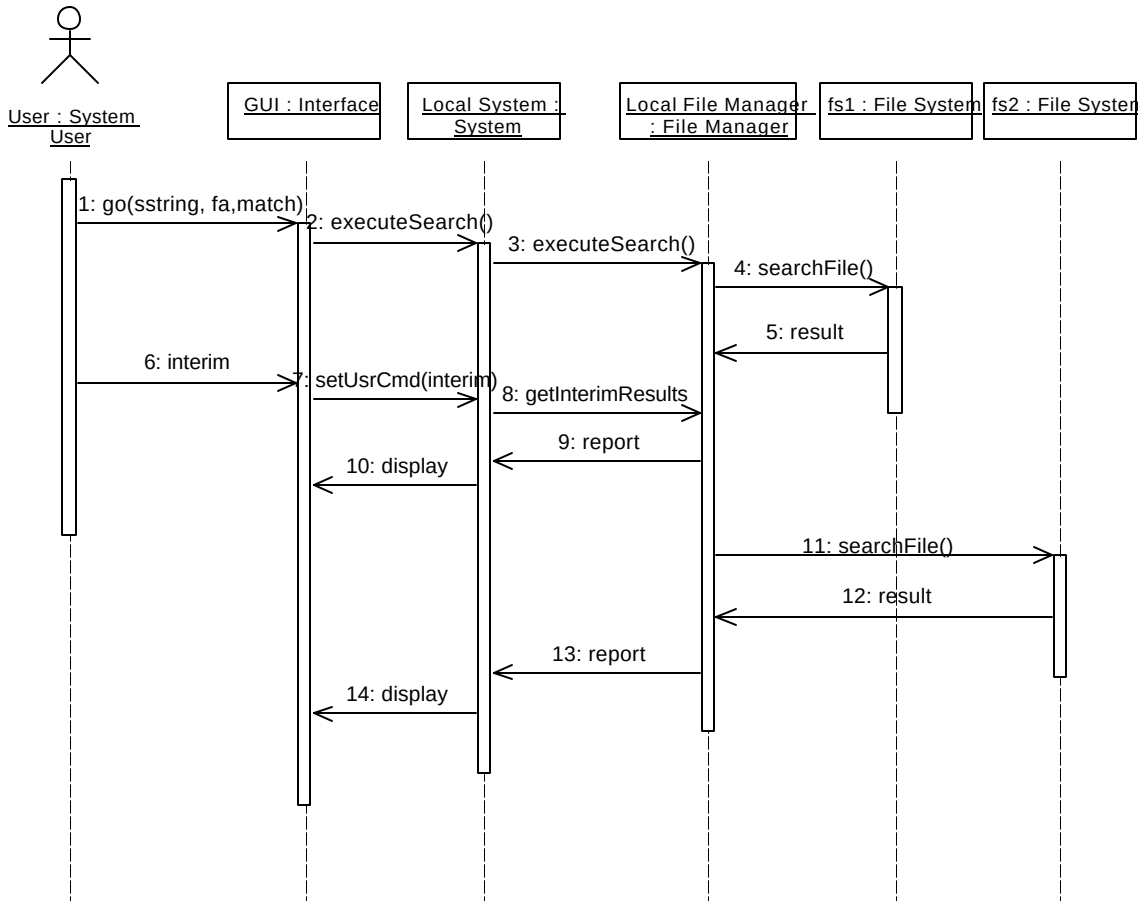

System User
Normal Search
Get Interim Results
Cancel Search

Sequence Diagrams:

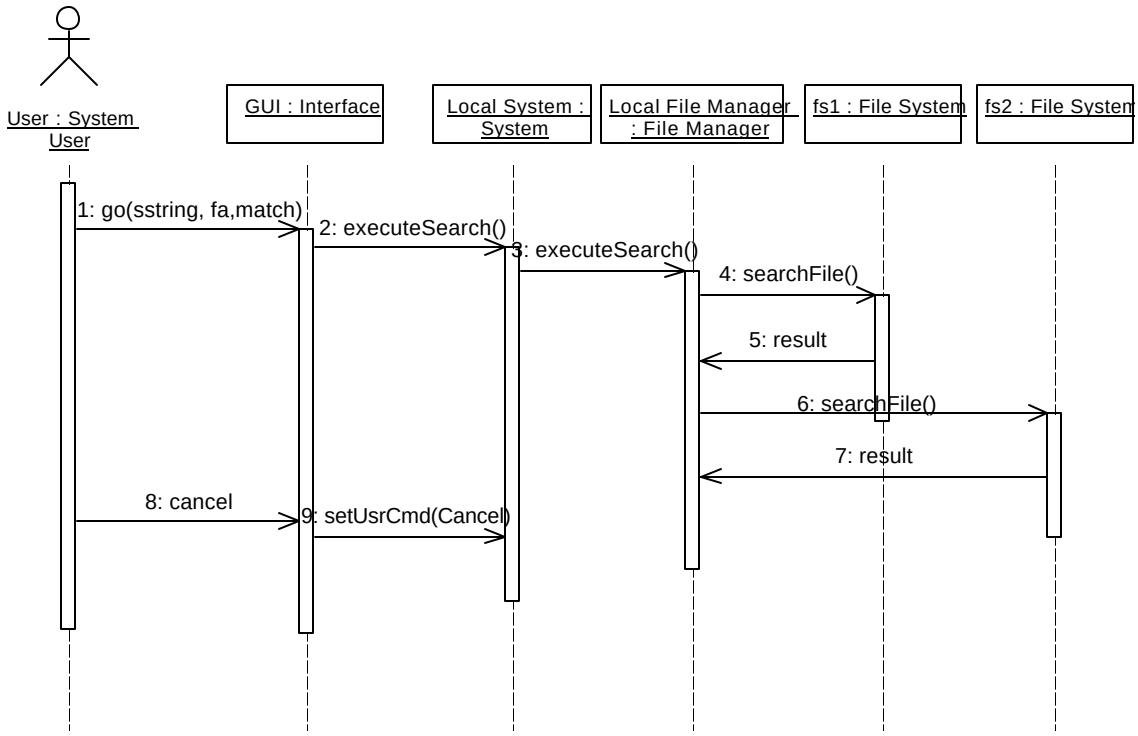
Normal Search:



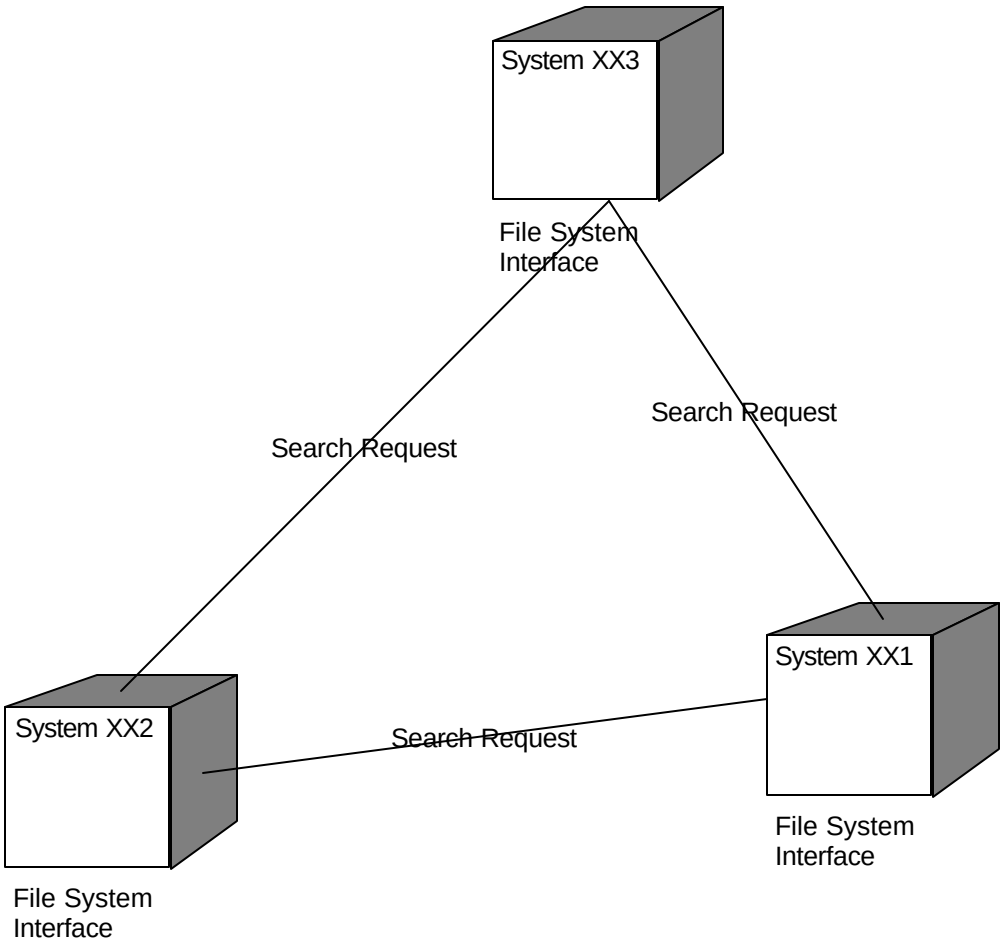
Get Interim Results:



Cancel Search:



Deployment View:



Data Dictionary

LOGICAL VIEW

1. Class name:
Search Request

Category: Logical View
Documentation:
A search request is initiated by the user. It will require the key word or phrase that is to be searched, whether or not an "exact" search is required, and the functional area of the search.

Associations:

<no rolename> : File Manager in association request list
<no rolename> : Interface in association generates
<no rolename> : Functional Area in association includes
<no rolename> : System in association Composed of
<no rolename> : File in association searches

Private Interface:
Attributes:
exactMatch : boolean
Boolean flag used to indicate whether or not the search request is for an "exact" search.

searchPhrase : string
The key word or phrase to be searched for.

2. Class name:
Report

Category: Logical View
Documentation:
A report is a collection of search request results.

Associations:

<no rolename> : Result in association reports
<no rolename> : Interface in association displays

Public Interface:
Operations:
add()
toString()

Private Interface:
Attributes:

results : collection of Result

3. Class name:
Result

Category: Logical View
Documentation:
A result is data collected after a search. It contains the location of the file, name of the document, and the number of occurrences of the key word or phrase.

Associations:

<no rolename> : Report in association reports

Public Interface:
Operations:
getLocation()
getFileName()
getOccurenceCount()
setLocation(string)
setFileName(string)
setOccurenceCount(integer)
toString

Private Interface:
Attributes:
location : string
The location of the file (full directory path).

fileName : string
The name of the file searched (full name with file extension).

occurrenceCount : integer
A count of the number of occurrences the key word or phrase was found in the file.

4. Class name:
Functional Area

Category: Logical View
Documentation:
A functional area describes the subject that the directory of files pertains to, as well as the type of information the search request is pertaining to.

Associations:

<no rolename> : Search Request in association includes

<no rolename> : File Manager in association pertains to

Private Interface:
Attributes:
name
The name that describes the
functional area.

5. Class name:
File

Category: Logical View
Documentation:
A file on a computer system

Associations:

<no rolename> : Search Request in association searches
<no rolename> : File List in association Assigned to

6. Class name:
System

Category: Logical View
Documentation:
System is the overall content search system.

Associations:

<no rolename> : Interface in association Composed of
<no rolename> : Search Request in association Composed
of
<no rolename> : File Manager in association Composed of

Private Interface:
Attributes:
userCmd

7. Class name:
Interface

Category: Logical View
Documentation:
The interface is the gui to the overall system
providing the user access to searching remote systems.
When a request is made, a Search Request is generated,
which is used for the search.

Associations:

<no rolename> : Report in association displays

```
            <no rolename> : Search Request in association generates
            <no rolename> : System in association Composed of

      Public Interface:
         Operations:
                     actionPerformed

      Private Interface:
         Attributes:
                     txtSearchString : textfield
                     txtFunctionalArea : textfield
                     chbx : checkbox
                     go : button
                     cancel : button
                     interim : button


  8.     Class name:
         File Manager

      Category:        Logical View
      Documentation:
         A File Manager maintains a list of file systems for the
         overall system.

      Associations:

            <no rolename> : Search Request in association request
list
            <no rolename> : Functional Area in association pertains
to
            <no rolename> : System in association Composed of
            <no rolename> : File System Int in association Manages

      Public Interface:
         Operations:
                     executeSearch()
                     handleInterim

      Private Interface:
         Attributes:
                     parent
                     fileSystems


   9.    Class name:
         File System

      Category:        Logical View
      Documentation:
         A File System is the interface to searching files.

      Associations:
```

```
               <no rolename> : File List in association Assigned to
               <no rolename> : File System Int in association
interfaces

      Public Interface:
         Operations:
                        getFunctionalArea()
                        getListofFiles()
                        searchFile(string)
                        setListofFiles(vector)
                        showListofFiles()

      Private Interface:
         Attributes:
                        rootDir : string
                        listofFiles : container of strings
                        host : string


  10.    Class name:
          File List

      Category:        Logical View
      Documentation:
         A file list is a set of files that are in a particular
         directory

      Associations:

               <no rolename> : File in association Assigned to
               <no rolename> : File System in association Assigned to


  11.    Class name:
          File System Int

      Category:        Logical View
      Documentation:
         Interface for remote File System

      Associations:

               <no rolename> : File Manager in association Manages
               <no rolename> : File System in association interfaces

      Public Interface:
         Operations:
                        getFunctionalArea()
                        getListofFiles()
                        searchFile(string)
                        setListofFiles(vector)
                        showListofFiles()
```

1. Association:
searches

Documentation:
When a search request is generated, it is intended to search a set of files that are associated with a specific functional area.

Role:
Class: File
Cardinality / Multiplicity: n

Role:
Class: Search Request
Cardinality / Multiplicity: 1

2. Association:
reports

Documentation:
A collection of results are collected in a report

Role:
Class: Result
Cardinality / Multiplicity: n

Role:
Class: Report
Cardinality / Multiplicity: 1

3. Association:
includes

Documentation:
A Search Request includes a Functional Area for the search

Role:
Class: Search Request
Cardinality / Multiplicity: n

Role:
Class: Functional Area
Cardinality / Multiplicity: 1

4. Association:
generates

Documentation:
The Interface generates Search Requests when a user requires a information

Role:
Class: Search Request
Cardinality / Multiplicity: 0..n

Role:
Class: Interface
Cardinality / Multiplicity: 1

5. Association:
displays

Documentation:
After a search is complete, the Interface displays a Report concerning the results of the search.

Role:
Class: Report
Cardinality / Multiplicity: 0..n

Role:
Class: Interface
Cardinality / Multiplicity:

6. Association:
Composed of

Documentation:
The System is composed of a user interface (Interface), user defined requests (Search Requests), and a set of known files (File Manager).

Role:
Class: System
Cardinality / Multiplicity: 1

Role:
Class: Interface
Cardinality / Multiplicity: 1

7. Association:
Composed of

Documentation:
The System is composed of a user interface (Interface), user defined requests (Search Requests), and a set of known files (File Manager).

Role:
Class: System
Cardinality / Multiplicity: 1

Role:
Class: Search Request
Cardinality / Multiplicity: 0..n

8. Association:
Assigned to

Documentation:
A File System is contains a File List, or collection of files.

Role:
Class: File System
Cardinality / Multiplicity: 1

Role:
Class: File List
Cardinality / Multiplicity: 1

9. Association:
Assigned to

Documentation:
Files are assigned to a File List

Role:
Class: File List
Cardinality / Multiplicity: 1

Role:
Class: File
Cardinality / Multiplicity: 0..n

10. Association:
Manages

Documentation:
The File Manager manages a set of interfaces to the file systems throughout the network

Role:
Class: File Manager
Cardinality / Multiplicity: 1

Role:
Class: File System Int
Cardinality / Multiplicity: 0..n

11. Association:

request list

Role:
Class: File Manager
Cardinality / Multiplicity:

Role:
Class: Search Request
Cardinality / Multiplicity:

12. Association:
Composed of

Documentation:
The System is composed of a user interface (Interface), user defined requests (Search Requests), and a set of known files (File Manager).

Role:
Class: System
Cardinality / Multiplicity: 1

Role:
Class: File Manager
Cardinality / Multiplicity: 1

13. Association:
pertains to

Documentation:
A File Manager manages a set of files that pertain to the Functional Area

Role:
Class: File Manager
Cardinality / Multiplicity:

Role:
Class: Functional Area
Cardinality / Multiplicity:

14. Association:
interfaces

Documentation:
File System Int is the Remote Method Invocation interface for File System.

Role:
Class: File System
Cardinality / Multiplicity: 0..n

Role:
Class: File System Int
Cardinality / Multiplicity: 1

USE CASES

1. Class name:
System User

Category: Use Case View
Documentation:
A system user is defined to be an employee of the organization that requires information regarding the location of information throughout the organization's network.

Stereotype: Actor

Associations:

<no rolename> : Normal Search in association
<no rolename> : Cancel Search in association
<no rolename> : Get Interim Results in association

1. Use Case name:
Normal Search

Category: Use Case View
Documentation:
The typical scenario for searching the network.

Associations:

<no rolename> : System User in association

2. Use Case name:
Get Interim Results

Category: Use Case View
Documentation:
The scenario requiring the system to return an intermediate set of results to the user

Associations:

<no rolename> : System User in association

3. Use Case name:
Cancel Search

Category: Use Case View
Documentation:
The scenario in which the user decides to abort the search.

Associations:

<no rolename> : System User in association

SEQUENCE DIAGRAMS

1. Object name:
User

Class: System User

2. Object name:
GUI

Class: Interface

3. Object name:
Local System

Class: System

4. Object name:
Local File Manager

Class: File Manager

5. Object name:
fs1

Documentation:
Local or remote File System

Class: File System

6. Object name:
fs2

Documentation:
Local or remote File System

Class: File System

DEPLOYMENT MODEL

1. Processor name:
System XX3

Documentation:
A system contains a search interface, and a file system. The file system is registered with an RMI registry.

The file manager, when initiated, accesses each of the remote file systems through the RMI registry.

Characteristics:
System XX3 is a Spark 10
IP Address: XXX.X.X.XXX
Host Name: XXX3

Processes: File System
Interface

2. Processor name:
System XX2

Documentation:
A system contains a search interface, and a file system. The file system is registered with an RMI registry.

The file manager, when intitiated, accesses each of the remote file systems through the RMI registry.

Characteristics:
System XX2 is a PC/Windows 2000
IP Address: XXX.X.X.XXX
Host Name: XXX2

Processes: File System
Interface

3. Processor name:
System XX1

Documentation:
A system contains a search interface, and a file system. The file system is registered with an RMI registry.

The file manager, when intitiated, accesses each of the remote file systems through the RMI registry.

Characteristics:
System XX1 is a Spark 10
IP Address: XXX.X.X.XXX
Host Name: XXX1

Processes: File System
Interface

1. Process name:
   File System

   Documentation:
      This process is used for searching the local file
      system for specified strings.

2. Process name:
   Interface

   Documentation:
      This thread manages a user interface for collecting
      requests and presenting results.

E.3 Software Development Questionnaire

Analysis and Design Representation Understanding Questionnaire

Below are 15 questions related to the packet of analysis and design models for a software system. Please take a few minutes to review the packet of models and data dictionary. After reviewing the packet, answer the following questions to the best of your ability. You may use the packet for reference as necessary.

---

1. Which set of models did you review? MaSE (agent-oriented) or Booch (object-oriented)

2. Are you familiar with this methodology? Yes or No

3. Have you ever used this methodology to develop software systems? Yes or No

4. List other methodologies you have used to develop software systems (please rank in order of familiarity, or use, from most to least):

1) _____
2) _____
3) _____
4) _____

5. On the continuum below, rate your general understanding of the models:

I got it!	I think I understand what's going on	I understand some of the models and understand the big picture	It looks familiar but I cannot determine what is going on	I do not understand these models
4	3	2	1	0

6. Which statement best describes the overall objective of the system described in the packet?

a) The system provides an interface to the user in order to find data files on a computer system that are relevant to a user-specified text string. ***(1)***
b) The system provides an interface to the user in order to find data files on a network of computer systems that are relevant to a user-specified string. ***(2)***
c) The system provides an interface to a network of computer systems within an organization allowing users to find data files that contain a specified string and related to a specified area of responsibility. ***(3)***
d) The system provides an interface to a network of computer systems within an organization allowing users to find data files relevant to a specified topic and related to a specified area of responsibility. Additionally, the system is capable of retrieving the file so the user can review the particular data. ***(2)***

7. Can you identify the incorporation of an existing legacy system? Yes ***(0)*** or No ***(1)***
If yes, where _____

8. Are previously designed components identifiable? Yes ***(0)*** or No ***(1)***
If yes, where _____

9. How many hardware systems have been identified for the implementation? ___ ***(1)***

10. What information can be identified regarding the hardware system? (Check *all* that apply) ***(6)***

___ Computer Name	___ Location of Computer	___ Type of Processor
___ Address	___ Number of Processes Assigned	___ Connectivity Properties

11. How many different operating systems have been identified for the implementation? _____ ***(1)***

12. Identify the input the system requires of the user. (Check *all* that apply) ***(6)***

___ Search String	___ Functional Area	___ Time limit to Search
___ Result format	___ Systems to be Searched	___ Limit Results to the top # of results

13. Identify the output of the system. (Check *all* that apply) ***(6)***

___ Name of the File	___ A copy of the File
___ Location of the File	___ Date and Time of File's last modification
___ File originator/creator	___ Number of times the Search String appears

14. Identify the options presented to the user. (Check *all* that apply) ***(5)***

___ Request for intermediate results	___ Repeat Search	___ Cancel the Search
___ Limit number of results returned	___ Template for result presentation	

15. Were you able to visualize the interface from any of the models? Yes ***(0)*** or No ***(1)***

Please verify that you have answered all the questions to the best of your ability.

E.4 Questionnaire Results (Tabulated Summary)

Question	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1)																		
MaSE	X	X	X	X	X	X	X	X	X	X								
Booch											X	X	X	X	X	X	X	X
2)	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y
3)	Y	Y	N	N	N	N	N	Y	Y	Y	Y	Y	N	N	Y	Y	N	Y
4)																		
(1)	5	6	6	6	5	8	5	6	6	2	3			2	1	2		1
(2)	6	5	5		6	7	6		5		4							
(3)						3			2									
(4)																		
5)	4	3	2	4	3	3	3	4	4	2	2	4	3	3	4	3	3	3
6)	B	B	B	D	C	B	C	B	B	B	C	C	C	D	C	C	C	C
7)	N	N	N	Y	N	N	N	N	N	N	N	N	N	Y	N	Y	Y	Y
8)	N	N	N	N	N	Y	N	N	N	N	Y	N		N	N	N	N	N
9)	3	3	10	3	3	3	3	3	3	3	2	3	3	2	3	3	3	2
10)																		
Computer Name	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X
Address	X	X	X	X	X	X	X	X		X	X	X		X	X	X		
Location				X								X						
Number	X	X				X	X		X	X		X		X	X		X	
Type		X		X		X	X			X		X	X		X	X	X	
Connectivity	X		X	X		X	X	X	X	X							X	
11)	2	2	2	2	1	2	2	2	0	2	2	2	1	3	2	2	2	2
12)																		
Search String	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Result Format																		
Functional Area	X						X				X	X	X	X	X	X	X	X
Systems				X														
Time				X														
Limit																		

13)																		
Name	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X	X
Location	X		X	X	X				X	X		X	X	X	X	X	X	X
Creator																		
Copy																		
Date/Time																		
Occurrences				X			X			X		X	X	X	X		X	X
14)																		
Intermediate	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X
Limit					X													
Repeat									X							X		
Template															X			
Cancel	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X
15)	N	N	Y	Y	Y	N	Y	N	Y	N	N	Y	Y	Y	Y	Y	Y	Y

Other Methodologies (Question 4)

1 – Structured

2 – Ad-Hoc

3 – UML

4 – AWL

5 – Functional

6 – Object-Oriented

7 – Z

8 – OMT

Bibliography

1. Bailey, Elizabeth, et al. "Practical Software Measurement – A Foundation for Objective Project Management." Office of the Under Secretary of Defense for Acquisition and Technology (April 1998).
2. Booch, Grady. Object-Oriented Analysis and Design with Applications. Redwood City CA: The Benjamin/Cummings Publishing Company, 1994.
3. Durfee, Edmund H. "Distributed Problem Solving and Planning," in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Gerhard Weiss ed. Cambridge Mass: MIT Press, 1999.
4. Fichman, Robert G. and Chris F. Kemerer. "Object-Oriented and Conventional Analysis and Design Methodologies," IEEE Computer p. 22-39 (October 1992).
5. Firth, Robert et al. "A Classification Scheme for Software Development Methods," Software Engineering Institute Technical Report 87-TR-41 (Nov 1987).
6. Goicoechea, Ambrose Don R. Hansen, and Lucien Duckstein. Multiobjective Decision Analysis with Engineering and Business Applications. New York: John Wiley & Sons, 1982.
7. Guessoum, Zahia and Jean-Pierre Briot. "From Active Objects to Autonomous Agents," IEEE Concurrency Vol 7 No3 : p 68-76 (July-Sept 1999).
8. Hendler, James. "Control of Agent-Based Systems (CoABS)," Excerpt from unpublished document, n. pag. http://dtsn.darpa.mil/iso/programtemp.asp?mode=126. 24 April 2000.
9. Huhns, Michael and Larry Stephens. "Multiagent Systems and Society of Agents" in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Gerhard Weiss ed. Cambridge Mass: MIT Press, 1999.
10. IEEE Standards Collection: Software Engineering, IEEE Standard 610.12-1990, IEEE, 1993.
11. Iglesias, C. A., M. Garijo, and J.C. Gonzalez. "A Survey of Agent-Oriented Methodologies," in Intelligent Agents V – Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), Lecture Notes in Artificial Intelligence. Heidelberg: Springer-Verlag, 1998.
12. Jennings, N.R. "On Agent-based Software Engineering," Artificial Intelligence: Vol 117 (2000), p 277-296.
13. Jennings, N.R. "Agent Oriented Software Engineering." Proc. 12th Int Conf on Industrial and Engineering Applications of AI, Cairo, Egypt, 4-10. (Invited paper) n. pag. http://www.ecs.soton.ac.uk/~nrj/pubs.html
14. Jennings, N. R. "Agent-Based Computing: Promise and Perils," in Proceedings of the 16th Joint Conference on Artificial Intelligence (IJCAI-99). Stockholm, Sweeden, p. 1429-1436 (1999).
15. Jennings, N.R. and M. J. Wooldridge. "Agent-Oriented Software Engineering." In Handbook of Agent Technology, ed. J. Bradshaw, AAI/MIT Press (To Appear).
16. Karpak, Birsen and Stanley Zionts eds. Multiple Criteria Decsion Making and Risk Analysis Using Microcomputers. Berlin: Springer-Verlag, 1989.
17. Kelley, Jay W. Air Force 2025. 2025 Support Office, Air University, Air Education and Training Command. Air University Press, August 1999.
18. Kirkwood, Craig W. Strategic Decision Making: Multiobjective Decision Analysis with Spreadsheets. Belmont CA: Wadsworth Publishing, 1997.

19. Kobryn, Cris. “Modeling Components and Frameworks with UML,” Communications of the ACM Vol 43: No 10 p. 31-38 (October 2000).

20. Lacey, Timothy H. A Formal Methodology and Technique for Verifying Communication Protocols in a Multi-Agent Environment. AFIT/GCS/ENG/00M-12. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

21. Lane, Thomas G. “Guidance for User Interface Architecture,” in Software Architecture: Perspectives on an Emerging Discipline. Mary Shaw ed. Upper Saddle River NJ: Prentice Hall, 1996.

22. Lawlis, Patricia K. “Guidelines for Choosing a Computer Language: Supporting the Visionary Organization” 2nd Ed. http://sw-eng.falls-church.va.us/AdaIC/docs/reports/lawlis/content.html (August 1997).

23. Lesser, Victor R. "Cooperative Multiagent Systems: A Personal View of the State of the Art," IEEE Transactions on Knowledge and Data Engineering Vol 11, No 1: p. 133-142 (Jan-Feb 1999).

24. “MESSAGE: Methodology for Engineering Systems of Software Agents – Initial Methodology.” EURESCOM Participants in Project P907-GI, July 2000.

25. Muller, Pierre-Alain. Instant UML. Birmingham UK: Wrox Press, 1997.

26. Noe, Penelope A. A Structured Approach to Software Tool Integration. AFIT/GCS/ENG/99M-14. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1999.

27. Nwana, Hyacinth et al. “ZEUS: A Toolkit and Approach for Building Distributed Multi-Agent Systems,” in Proceedings of the Third Annual Conference on Autonomous Agents. Seattle WA, (May 1999).

28. Odell, James. “Objects and Agents: How do they differ?” Unpublished document, n. pag. http://www.jamesodell.com/publications.html September 1999.

29. O’Malley, Scott, Athie Self, and Scott DeLoach. “Comparing Performance of Static versus Mobile Multiagent Systems,” in Proceedings of NAECON 2000. Dayton OH, (Oct 2000).

30. Orfali, Robert et al. The Essential Distributed Objects Survival Guide. New York: John Wiley & Sons, Inc. 1996.

31. Pressman, Roger S. Software Engineering: A Practitioner’s Approach. New York: McGraw-Hill, 1997.

32. Robinson, David J. A Component Based Approach to Agent Specification. AFIT/GCS/ENG/00M-22. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

33. Ryan, Michael. Commanders NOTAM 00-5, July 2000.

34. Sametinger, Johannes. Software Engineering with Reusable Components. New York: Springer, 1997.

35. Shelton, Henry. Joint Vision 2020. Joint Staff: Pentagon, 2000.

36. Shen, Weiming and Douglas Norrie. “Agent-Based Systems for Intelligent Manufacturing: A State-of the-Art Survey,” Knowledge and Information Systems, an International Journal. Vol 1, Num 2. p 129-156 (1999).

37. Szyperski, Clemens. Component Software, Beyond Object-Oriented Programming. New York: ACM Press, 1999.

38. Wood, Bill, Richard Pethia, Lauren Roberts Gold, and Robert Firth. "A Guide to the Assessment of Software Development Methods," Software Engineering Institute Technical Report 88-TR-8, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, July 1988.

39. Wood, Mark F. Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems. AFIT/GCS/ENG/00M-26. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

40. Wooldridge, M. "Intelligent Agents," in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Gerhard Weiss ed. Cambridge Mass: MIT Press, 1999.

41. Wooldridge, M.J. and N.R. Jennings. "Pitfalls of Agent-Oriented Development," Proceedings 2nd International Conference on Autonomous Agents (Agents-98), Minneapolis MN USA, p. 385-391 (1998).

42. Wooldridge, M.J., N.R. Jennings and D. Kinney. "The Gaia Methodology for Agent-Oriented Analysis and Design," International Journal of Autonomous Agents and Multi-Agent Systems, vol 3, (2000) p. 1-27.

43. Yourdon, Edward. Modern Structured Analysis. Englewood Cliffs NJ: Yourdon Press, 1989.

VITA

First Lieutenant Scott Alan O'Malley was born in Allegan, Michigan. He graduated from Indianola High School in Indianola, Iowa in June 1993. After completing his Bachelor of Science degree in Computer Science in May 1997 at the University of Iowa, he was commissioned through the Air Force Reserved Officer Training Corps (ROTC).

Lieutenant O'Malley's first assignment was at Hickam AFB, Hawaii as an executive officer for the 15th Support Group in September 1997. In June 1998, he was assigned as a communications-information officer in the 15th Communications Squadron at Hickam AFB, where he held a number of positions throughout the squadron while participating in a program called Aerospace Communications Expertise (ACE). In August 1999, he entered the Graduate School of Engineering and Management's Computer Science program, Air Force Institute of Technology. Upon graduation, he will be assigned to the United States Strategic Command at Offutt AFB, Nebraska.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
20-03-2001	Master's Thesis	April 2000 - March 2001

4. TITLE AND SUBTITLE
SELECTING A SOFTWARE ENGINEERING METHODOLOGY USING MULTIOBJECTIVE DECISION ANALYSIS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Scott Alan O'Malley

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 P Street, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/01M-08

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr. Robert L. Herklotz
AFOSR/NM, Program Manager: Software and Systems
801 N. Randolph Street, Room 732
Arlington VA 22203-1977
(703) 696-6565

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES
SCOTT A. DELOACH, Major, USAF, Ph.D., Assistant Professor, (937)255-3636 x4581, scott.deloach@afit.edu

14. ABSTRACT

With the emergence of agent-oriented software engineering methodologies, software developers have a new set of tools to solve complex software requirements. One problem software developers face is to determine which methodology is the best approach to take to developing a solution. A number of factors go into the decision process. This thesis defines a decision making process that can be used by a software engineer to determine whether or not a software engineering approach is an appropriate system development strategy. This decision analysis process allows the software engineer to classify and evaluate a set of methodologies while specifically considering the software requirement at hand.

The decision-making process is developed on a multiobjective decision analysis technique. This type of technique is necessary as there are a number of different, and sometimes conflicting, criterions. The set of criteria used to base the decision was derived from literature sources and validated by an opinion survey conducted to members of the software engineering community. After developing the decision-making framework, a number of case studies are examined.

15. SUBJECT TERMS
Software Engineering, Specifications, Methodology, Decsion Analysis

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Maj Scott DeLoach
U	U	U	UU	254	19b. TELEPHONE NUMBER *(Include area code)* (937)255-3636 x 4581