UNIVERSITY OF CALIFORNIA

SANTA CRUZ

The Development of Representation-Based Methods for Knowledge Engineering of Computer Networks

A dissertation submitted in partial satisfaction
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

TECHNOLOGY AND INFORMATION MANAGEMENT

by

Tyler Munger

September 2016

The Dissertation of Tyler Munger
is approved:

Subhas Desa, Chair

Patrick Mantey

Arnav Jhala

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

The Development of Representation-Based Methods for Knowledge Engineering of Computer Networks

by

Tyler Munger

Enterprises that provide technical support for network engineering products and technologies are continuously accumulating terabytes of unstructured customer data in the form of service requests, device logs, bug reports, and network configurations. A large body of work in knowledge engineering (artificial intelligence, machine learning, data mining, and information retrieval) has been developed and applied within the context of important application areas such as expert systems, search engines, and more recently recommender systems. However, a relatively small body of knowledge engineering (KE) work has addressed methods and application of KE to the design and development of engineering systems and products. Furthermore, a relatively small proportion of the KE research applied to engineering systems has addressed all of the following three issues: 1) the structuring of the engineering subject-matter domains to which the theory is applied, 2) the proper integration of this domain structure with analytical KE methods, and 3) the KE software necessary to support the design and development of engineering systems.

This thesis addresses the theories, application, and implementation of knowledge engineering methods for using this collected data to improve product design, development, and delivery. To address the aforementioned three issues we have formulated a cognitive science-based representation framework for problem-solving, consisting of five sequential layers, or stages, which enables integration of diverse domains such as machine learning, product design and development, software engineering, and the statistical design of experiments.

To demonstrate and test our theories we have applied this representation-based framework, called the Integrated Meta-Representational Model (IMRM), to solve three important product design, development, and delivery problems related to computer networks. The first problem involved combining computer network domain knowledge with analytical methods from data mining and time-series analysis in order to monitor and assess the quality of a computer network security product. The second problem involved predicting whether or not an incoming customer support case should be escalated in priority in order to be resolved in a timely and cost-effective manner. For this problem we used a statistical Design of Experiments approach to optimizing the machine learning model for predicting whether or not a service request needs to be escalated. The third problem involved the development of a Knowledge Engineering Software Product for supporting the extraction of problem-solution pairs from customer service requests in order to create new computer network products and services. The work concludes by indicating how the Integrated Meta-Representational Model can be used to solve even more complex problems, involving the integration of all the core activities in engineering: design, analysis, experimentation, and prototyping/manufacturing.

DEDICATION

This thesis is dedicated to my parents, Don and Rebecca Munger. Without their love and support, this thesis would never have been possible.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Subhas Desa for giving me the opportunity to work on this thesis. I have had the privilege of working with Subhas as both an undergraduate and graduate student, and I could not have asked for a better mentor. I am very grateful for the knowledge and wisdom he has shared with me. Subhas is also responsible for one of the core ideas in this thesis, the Integrated Meta-Representational Model, which provided the theoretical framework for formalizing the integration of methods and tools from machine learning, product design and development, software engineering, and the statistical design of experiments.

I would like to thank the Smart Services Technology Group at Cisco Systems for providing me the opportunity to develop and test the ideas in this thesis within the context of real-world Knowledge Engineering problems. In particular, I owe special thanks to Sri Ramachandran, Chris Wong, and Jiabin Zhao for their support and collaboration. I would also like to thank the graduate students who have helped me over the years and contributed to the work in this thesis. Danny Core assisted in the development several key components of the knowledge engineering software product described Chapter 6. Shuang Wang, who had performed earlier work on the service request escalation problem described in Chapter 5, provided advice when I first started working on this problem and helped with obtaining the necessary data-sets.

None of this work would have been possible without the support of the Network Management Operations Lab. In particular, I would like to thank Patrick Mantey and Brad Smith who have been invaluable sources of knowledge, encouragement, and support.

Finally, I would like to thank Patrick Mantey and Arnav Jhala for serving as members of my thesis committee. I greatly appreciate the time they have spent reading drafts and providing feedback.

1 Introduction

A large body of work in knowledge engineering (artificial intelligence, machine learning, data mining, and information retrieval) has been developed and applied within the context of important application areas such as expert systems, search engines, and more recently recommender systems. However, a relatively small body of work has addressed the development of methods and tools for the application of knowledge engineering to the design and development of engineering systems and products. For the application of knowledge engineering to a particular engineering domain, such as computer networks, to yield meaningful and reliable (robust) results, the subject-matter of the domain most first be structured for the purposes of extracting knowledge.

Furthermore, only a relatively small proportion of the knowledge engineering research applied to engineering systems has addressed all of the following three areas: 1) the a priori structuring of the engineering subject matter domains to which the theory is applied, 2) the integration of this domain structure with analytical knowledge engineering methods, and 3) the design and development of software products to automate processes used by engineering teams in the design and development of engineering systems.

This thesis addresses the theories, implementation, and application of knowledge engineering methods to improve product design, development, and delivery in engineering domains. The purpose of this chapter is to introduce knowledge engineering within this context, outline some of the research issues involved, and describe our key contributions to these research issues.

To this end, the chapter is organized into four parts as follows. Section 1.1 provides a brief background section that motivates knowledge engineering in the product design, development, and delivery context, and differentiates it from existing work in data mining and machine learning. Section 1.2 highlights an important gap in existing knowledge engineering work, and develops the research issues related to applying knowledge engineering methods in complex technical domains. Following from these research issues, Section 1.3 summarizes the key theoretical and applied research contributions of the work. Section 1.4 concludes the chapter by outlining the overall structure of the thesis.

1.1 Background

Most enterprises, technology and otherwise, are routinely collecting massive amounts of information from customers. In general, the customer data being collected falls into two distinct categories or groups: structured data resulting from customer transactions such as sales, and unstructured data from customer interactions such as customer support.

The extraction of knowledge from customer transaction data is a well-defined problem that has been studied extensively by the data mining community. It is relatively straightforward to solve these problems by directly applying standard data mining methods and tools. For example, consider the problem of mining customer sales data to determine the sets of products that customers frequently purchase together. Since sales data is usually recorded in a structured fixed-field format, meaningful patterns can typically be directly extracted using well-known techniques such as association analysis [Witten and Frank, 2005].

The extraction of knowledge from customer interaction data, however, is a far more complex and difficult task [Spangler and Kreulen, 2008]. Directly applying data mining methods rarely produces results that are applicable to enterprise business/technology activities such as product development. For example, consider the extraction of frequently encountered customer product problems from resolved technical support cases. In this case, the direct application of data mining techniques such as cluster analysis [Witten and Frank, 2005] is unlikely to produce results that are meaningful in the context of identifying common product problems.

The efficient extraction of knowledge from customer interaction data is of great interest to nearly all enterprises, technology and otherwise, as the collected data contains knowledge and insights that enable the design, development, and delivery of smarter, more customer-centric, products and services. Some examples of the potential application of knowledge engineering within the context of product design, development, and delivery (support) are as follows:

1. **Product Design**: Determining what functions and features need to be included in the next generation of products.
2. **Product Development**: Determining key product failure modes that need to be resolved during product development.
3. **Product Delivery**: Automating the detection and resolution of product related problems in service.

The knowledge engineering problem of extracting meaningful knowledge from customer interaction data is a complex multi-disciplinary problem with the following characteristics:

1. A complete and comprehensive solution to the problem requires the use of several subject-matter domains.
2. Each domain has its own representational language.
3. Each domain has, in general, its own unique set of methods and tools for addressing and resolving problems.
4. The methods and tools from the diverse subject-matter domains need to be integrated in order to solve the specified problem.

The combination of these characteristics requires a framework to organize and structure the domains involved in knowledge engineering, as well as the diverse methods and tools from these domains necessary for the problem in hand.

1.2 Research Issues

This thesis addresses two distinct problems. The first problem is the development of the framework discussed in the previous section for selecting and integrating the methods and tools from the multiple engineering domains. The theoretical research issues related to the development of this framework are listed below:

Theoretical Research Issue 1: What engineering subject matter domains, methods and tools are necessary when addressing a particular knowledge engineering problem?

Theoretical Research Issue 2: How should the necessary methods and tools be integrated into a comprehensive end-to-end process methodology for solving the knowledge engineering problem under consideration?

The second problem addressed in this thesis is the application of the developed framework to solve complex real-world knowledge engineering problems. This problem involves the following applied research issues:

Applied Research Issue 1: How should the necessary domain knowledge for solving a particular knowledge engineering problem be collected and structured in order to obtain meaningful and reliable results?

Applied Research Issue 2: How should the settings or parameters for the machine learning process be selected in order to produce the best predictive model with respect to the knowledge engineering problem under consideration?

Applied Research Issue 3: How should the appropriate software necessary for automating the processes used by engineers to solve a particular knowledge engineering problem be designed, developed, and implemented?

1.3 Research Contributions

This work makes two key sets of contributions to the knowledge engineering research issues described in the previous section. The first set of contributions are related to the theoretical framework that we have developed for layering any complex knowledge engineering problem so that the correct set of methods and tools can be applied to its solution. The second set of research contributions are related to the application of this theoretical framework to solve three important knowledge engineering problems in the computer networking domain.

1.3.1 Theory: Integrated Meta-Representational Model

To address the aforementioned research issues we have formulated a cognitive-science based framework, called the Integrated Meta-Representational Model (IMRM), consisting of five inter-related-functional problem-solving layers. This functional layering of a problem provides an abstract "subject-matter domain-neutral model" that then enables the selection of the appropriate subject matter domains and associated methods and tools for each layer of the representation. We apply the IMRM to organize and coordinate different activities involved in applying knowledge engineering to the design and development of engineering systems and products.

The integrated representational structure of the IMRM allowed us to answer the theoretical research issues 1 and 2:

Theoretical Research Issue 1: What engineering subject matter do-

mains, methods and tools are necessary when addressing a particular knowledge engineering problem?

The functional layering of the IMRM enables the selection of the necessary methods and tools from the domains of machine learning, product design and development, statistical design of experiments, and software engineering. A summary of the methods and techniques from the domains used in this works is provided in Chapter 2.

Theoretical Research Issue 2: How should the necessary methods and tools be integrated into a comprehensive end-to-end process methodology for solving the knowledge engineering problem under consideration?

Multi-disciplinary development is notably difficult because each subject matter domain has its own unique representational language, methods, tools, and techniques. Furthermore, these diverse methods and tools need to be properly integrated in order to provide a comprehensive and complete solution to the given problem. The IMRM provides a common functional representation-based language necessary for integrating methods across multiple domains into a unified development process capable of producing comprehensive solutions to the problem. A description of this integration is discussed in Chapter 3.

1.3.2 Representation-Based Models for Knowledge Engineering in the Computer Networking Domain

The second set of contributions are related to the application of the Integrated Meta-Representational Model (IMRM) to the three important real-world knowledge engineering problems in the computer network services application domain.

1. **Product Quality Monitoring and Assessment**

We applied the IMRM to the knowledge engineering problem of monitoring and assessing the quality of a specific computer network security device using historical customer service request data. In this problem we show how the IMRM enables the a priori structuring of the domain knowledge for a particular problem, in this case the key failure modes for the network security device, and the subsequent integration of this domain knowledge with analytical knowledge engineering methods. This problem, addressed in Chapter 4, is related to Applied Research Issue 1.

2. **Predicting Service Request Escalation**

   We applied the IMRM to the knowledge engineering problem of predicting which customer service requests would need to be escalated in priority. In this problem we show how the IMRM enables the analytical knowledge engineering method, in this case the machine learning model for predicting escalation, to be optimized or tuned in order to produce the best results for the problem under consideration. This problem, addressed in Chapter 5, is related to Applied Research Issue 2.

3. **Development of Knowledge Engineering Software Products**

   We applied the IMRM to the knowledge engineering problem of extracting problem-solution pairs from customer service requests in order to improve the development of network products and services. In this problem we show how the IMRM enables the rapid development of high-quality knowledge engineering software automation, in this case a knowledge engineering software product to improve the productivity of the engineering team. This problem, addressed in Chapter 6, is related to Applied Research Issue 3.

The representation-based models developed for each of these problems, described below, enables us to answer the following applied research issues:

Applied Research Issue 1: How should the necessary domain knowledge

for solving a particular knowledge engineering problem be collected and structured in a coherent, goal-oriented manner?

There are two distinct types of domain knowledge that are often necessary in order to solve product design, development, and delivery problems: organizational knowledge and engineering knowledge. Organizational knowledge includes organizational factors and work-processes. Engineering knowledge, on the other-hand, is based around engineering issues such as product functionality, product subsystems, and product failure modes

In order to properly capture and structure the organizational knowledge we have drawn upon two modeling tools from the well-known CommonKADS methodology [Schreiber, 1994] for developing expert systems: the organization model to provide a concise high-level view of organizational context, i.e. people, processes, and resources, and the Agent / Task Model to provide a single integrated model for representing work processes using tasks, agents, and the relationships between agents and tasks. A detailed description of the CommonKADS methodology is provided in Chapter 6.

In order to represent the engineering domain knowledge in product design, development, and delivery problems we have drawn upon three well-known methods from engineering design. The Function Structure is an abstract implementation or solution neutral representation of the functions and sub-functions of a product. The Functional Analysis System Technique (FAST) [Fox, 1993] relates product sub-functions to the product sub-systems that implement these sub-functions. The Failure Modes and Effects Analysis (FMEA) [Fox, 1993] identifies the potential types of failure modes for products sub-systems, the effect of these failures on product functionality, and prescribes corrective actions that should taken. A detailed description of the FAST and FMEA methods is provided in Chapter 4.

Applied Research Issue 2: How should the settings or parameters for

the machine learning process be selected in order to produce the best predictive model for the knowledge engineering problem under consideration?

A key step in many knowledge engineering problems is the selection of the values of the different parameters involved in the machine learning process. Examples of some of the parameters that need to be selected include: the features used for representing the data and the type of machine learning algorithm to be used. The process of determining the optimal set of parameters for a given application is typically addressed through time-consuming heuristics and ad-hoc experimentation. To address and resolve this issue, we have developed a simple and effective approach based on statistical Design of Experiments (DOE) for optimizing the parameters associated with the machine learning process. In the DOE approach [Taguchi and Konishi, 1987] we treat each external model parameter, including the learning algorithm, as an experimental factor. We then design a set of experiments using orthogonal arrays to efficiently explore the experimental space of model parameters and determine the settings that produce the optimal model with respect to a desired performance metric. A detailed description of the statistical DOE approach is provided in Chapter 5.

Applied Research Issue 3: How should the appropriate software necessary for automating the processes used by engineers to solve a particular knowledge engineering problem be designed, developed, and implemented?

The deployment of efficient and robust solutions to knowledge engineering problems requires custom software products that automate the work process of the engineers engaged in the solution of these problems. To this end, we draw upon the engineering design domain to provide formal methods for explicitly defining the user needs and exploring different design concepts in order to ensure that the end product is high-quality and low-cost. We have integrated four engineering design methods for the purposes of developing high-value knowledge engineering software products (KESPs). First, the House of Quality is used to correlate the user needs to technical specifications that can be used to develop the KESP. The function structure technique is then used to create a solution-neutral functional specification of the KESP. Next, we create a morphological matrix of solution principles in order to systematically explore the design space and generate alternative KESP design concepts. Lastly, a utility function is used to assess the design concepts so that a single high-value concept can be selected for further development. In addition, the development process for KESPs requires software engineering methods and tools, such as the Unified Modeling Language Schach [2008], in order to ensure that a reliable and easy to use KESP is delivered on time and within budget. A detailed description of the aforementioned engineering design and software engineering methods and tools is provided in Chapter 6.

1.4 Organization of the Thesis

The thesis is organized as follows: Chapter 2 develops the three knowledge engineering problems addressed in the thesis, formulates the overarching research issue, and surveys existing work in related domains. Chapter 3 develops the Integrated Meta-Representational Model (IMRM) for structuring complex knowledge engineering problems such as the three knowledge engineering problems described in Chapter 2. The implementation of the representation-based models for the three knowledge engineering problems of interest is then discussed in Chapters 4 (product quality monitoring and assessment), 5 (predicting service request escalation), 6 (development of knowledge engineering software products). Chapter 7 summarizes the key contributions of the work and then suggests future work related to the generalization of the representation-based approach developed in this thesis.

2 Problem Statement

The purpose of this chapter is to develop the problem statement for the creation of a theoretical framework that enables the selection and integration of methods from multiple engineering domains within the context of knowledge engineering problems. In order to motivate the need for this framework we use three complex knowledge engineering problems related to the design, development, and delivery of enterprise computer networking products. The formulation of these problems in Section 2.1 leads to a discussion of the inter-disciplinary nature of knowledge engineering in Section 2.2. Section 2.3 surveys related work in the engineering domains involved, and reveals a large number of useful methods and tools but a distinct lack of frameworks for systematically identifying and applying the necessary methods and tools for a particular problem. Based on this important need, section 2.4 defines the overall problem statement for the thesis and outlines the tasks involved.

2.1 Three Knowledge Engineering Problems in the Computer Networking Domain

This thesis addresses the following three problems in the computer networking domain:

1. **Product Quality Monitoring and Assessment**: use unstructured customer service request data to monitor and assess the quality of computer networking products in service.

2. **Predicting Service Request Escalation**: use historical customer service requests to determine whether or not a new customer service request should be escalated in priority to ensure that it is resolved in a timely manner.

3. **Development of Knowledge Engineering Software Products**: create software for automating the extraction of problem-solution pairs from customer service

requests in order to improve the productivity of the work-process of engineering product development teams.

The sub-sections below provide a brief description of each of the three problems and summarize the key issues involved.

2.1.1 Product Quality Monitoring and Assessment

Monitoring and assessment of the quality of products in service is an important feedback loop for product design, development, and delivery. Network service centers receive thousands of service requests every day on a wide range of customer problems, both hardware and software. Although readily available, customer service request data typically consists of free-form text problem descriptions provided by the customer, which makes it difficult to directly apply existing monitoring approaches, such as control charts, that are based on time-series analysis.

Each service request consists of an unstructured (free-form text) customer description of a failure mode that occurred and a time-stamp. The unstructured format of the problem description poses the following challenges: customers use different language to describe the same failure mode; computer-network specific-domain terminology; and duplicate or irrelevant information.

Monitoring and assessing product quality using service request data involves the following four sub-problems:

1. **Failure Modes Definition**: Determine the set of failure modes that we would like to monitor for the product of interest.

2. **Failure Mode Quantification**: Measure the daily occurrence of each failure mode in the collected service requests.

3. **Time-series Analysis**: Determine the nominal (expected) daily occurrence of each failure mode and the deviation of the actual daily occurrence from its nominal value.

4. **Quality Metrics Computation**: Define and compute a set of metrics for characterizing the occurrence of each failure mode.

Each of the four sub-problems, and the overall product quality monitoring and assessment problem, is described in detail in Chapter 4.

2.1.2 Predicting Service Request Escalation

Network service centers are responsible for providing customers with assistance for a wide range of product problems ranging from relatively simple issues, e.g. such as how to configure a particular feature, to severe failure modes that cause network downtime. After a service request is received by the customer support organization, it is routed to the team of support engineers best suited for resolving the customer's problem. This routing decision is based on number of factors including the type of product, technology area, and severity of the problem. For a small percentage of cases it is necessary to escalate the service request in priority and re-route it to a new support team in order to resolve the customer's problem in a timely manner. The escalation process after a service request has already been re-routed is typically very expensive; escalating a single service request typically costs several hundred thousand dollars and is resource intensive. Predicting which service requests are likely to be escalated and escalating them immediately after they are received by the service center would significantly reduce the cost associated with the escalation process.

The problem of predicting whether or not an incoming service request should be escalated is a standard binary classification problem. We need to create a machine learning classification model for mapping service requests to escalation labels (escalate or do not escalate). The machine learning process for creating this machine learning model has the following complications that need to be resolved:

1. There are number of well-known supervised machine learning algorithms (decision trees, support vector machines, etc.) that can be used to generate a classification function from historical data; however, the performance of these algorithms can vary significantly across different data-sets. We need to determine the best algorithm for the service request escalation problem.
2. The historical data-set of service requests is imbalanced, i.e. the majority of service requests do not require escalation.
3. The misclassification costs associated with each service request are not symmetrical. A service request that is not escalated when it should have been will have several times the cost when compared to a service request that is incorrectly escalated when it should have not been, with respect to both dollar cost and customer satisfaction to the organization.
4. Each service request contains over a hundred different features or attributes. Most of these features, e.g. customer address, are not relevant or even useful for predicting if a service request needs to be escalated.

The complete description of the predicting service request escalation problem is provided Chapter 5.

2.1.3 Development of Knowledge Engineering Software Products

Smart network devices, e.g. routers and switches, pro-actively detect potential problems and take corrective actions. The intelligence in these products comes from a set of diagnostic rules that are written by Network Knowledge Engineers (NKEs). In order to write these rules, the NKEs first search through the resolved customer service requests in order to extract problem-solution pairs for frequent customer problems.

The current, largely manual, process for extracting problem-solution pairs consists of two high-level activities: searching for relevant service requests to a particular problem, and reading the relevant service requests in order to extract problem-solution pairs. The NKEs currently use a search engine to perform keyword searches in order to locate relevant service requests, and a web-based viewer to read the service requests.

The NKEs' manual work process is very inefficient because of the following problems:

1. **Precision problem**: Keyword searches typically return a large number of search results that are time consuming to evaluate for relevance.
2. **Summary problem**: The service request documents do not include a clear description of the solution that resolved the customer's problem.
3. **Repetition**: The service request documents typically contain a large amount of repetition in the form of repeated email threads and redundant case notes.

The "precision" problem impacts the information retrieval aspects of the NKE work process and requires the NKEs to manually evaluate a large number of irrelevant search results. The "summary" and "repetition" problems impact the extraction of problem-solution pairs from relevant service requests. In particular, the long length of each service request (typically 30-50 pages of free-form text) in combination with the presence of irrelevant email threads, poorly formatted text, and duplicate content make reading each service request difficult and time consuming.

From the perspective of the NKEs, the KESP problem statement is as follows: develop a software product to automate the tedious and manual aspects of extracting problem-solution pairs. In particular, the NKEs would like the software product to make it easier to find relevant service requests for a particular product problem (the "precision" problem) and extract the problem-solution pairs from a given service request (the "summary", and "repetition" problems). From the perspective of the networking company, the KESP problem statement is as follows: develop a software product to improve the productivity of the NKEs and decrease the cost of developing problem-solution pairs. The complete description of the development of knowledge engineering software products problem is provided in Chapter 6.

2.2 The Engineering of Knowledge Engineering

It quickly becomes apparent that addressing the aforementioned knowledge engineering problems requires more than one engineering domain. First, each problem involves domain knowledge about computer networking products and services which needs to be modeled and incorporated into the overall solution to the problem. Second, because the problems involve a combination of structured and unstructured data we need to use different kinds of analytical techniques. For example, for the monitoring and assessment problem we need to use machine learning to translate the unstructured customer service request data into time-series data that can be processed using techniques from time-series analysis. Third, these problems require the development of software artifacts that needs to be embedded into the overall organization that designs and develops the computer networking products and services.

Before we discuss the engineering domains required to solve these problems in more detail, however, it is useful to first describe the nature of the activities involved in engineering in general and how these activities relate to the three problems under consideration.

There are a central core of activities that are general to all engineering problem regardless of the domain or application area. In this work we distinguish between the following four fundamental engineering activities that together form the basis for engineering as a discipline:

1. **Design**: the development of a concept (form) for the engineering system to satisfy a set of (customer) needs. [Pahl and Beitz, 1996]
2. **Analysis**: the mathematical modeling of the important aspects of engineering systems. Ver Planck and Teare [1954]
3. **Experimentation**: the generation of information to guide design and prototyping / manufacturing decisions. [Srinagesh, 2006]
4. **Prototyping/Manufacturing**: the building and testing of prototypes for various purposes including "proof-of-concept" and production. [Pahl and Beitz, 1996] [Schach, 2008]

Each of the three knowledge engineering problem instances described in the previous section—quality monitoring and assessment, predicting service request escalation, and development of knowledge engineering software products—involve one or more of the fundamental engineering activities. Table 2.1 below shows the decomposition of each problem based on the four engineering activities. The shading associated with each circle indicates the intensity of the activity involved.

In general any particular problem will have a primary focus with respect to the four engineering activities. For example, the primary activity for the product quality monitoring and assessment problem is analysis. This primary activity is indicated in Table 2.1 by a fully shaded circle.

In addition to the focus area, the complex nature of these problem also requires other aspects of engineering. Each of these secondary activities are shown as a half-filled circle in 2.1. For example, the product quality monitoring and assessment problem

	Design	Analysis	Experiments	Prototyping Manufacturing
Quality Monitoring and Assessment	◑	●	○	◑
Service Request Escalation	○	◑	●	○
Development of Knowledge Engineering Software Products	●	○	◑	◑

Table 2.1: Analysis of the three knowledge engineering problems with respect to the four engineering activities

also involves design activities where we are modeling computer networking products and prototyping and manufacturing activities where we are developing software automation.

2.3 Literature Survey of Knowledge Engineering Domains

In the previous section we established that design, analysis, experimentation, and prototyping/manufacturing are general activities that occur across a wide range of engineering applications. As a result, there is a large body of work in engineering that addresses issues related to design, analysis, experimentation, and prototyping/manufacturing. In the context of knowledge engineering problems the following four engineering domains contain methods and tools that are useful for addressing these activities:

1. **Machine Learning (Analysis)** : extracting patterns and learning predictive models from data ([Witten and Frank, 2005], [Hastie et al., 2009]).
2. **Software Engineering (Design, Prototyping / Manufacturing)**: designing and implementing reliable and easy to use software systems [Schach, 2008].
3. **Product Design and Development (Design, Analysis)**: designing high-quality products that satisfy user needs ([Fox, 1993], [Pahl and Beitz, 1996]).

4. **Experimental Statistics (Experimentation)**: the use of statistical methodologies in the analysis and design of experiments ([Fisher, 1935], [Taguchi and Konishi, 1987], [Srinagesh, 2006]).

A detailed review of literature of the methods from each of these engineering domains is provided in Chapters 4, 5, and 6. Chapter 4 provides a detailed survey of work in machine learning and product design and development within the context of the product quality monitoring and assessment problem. Chapter 5 provides a detailed survey of work in machine learning and experimental statistics within the context of the problem of predicting service request escalation. Chapter 6 provides a detailed survey of work in product design and development and software engineering within the context of the problem of developing knowledge engineering software products.

2.4 Thesis Problem Statement

The rapid development of high-quality solutions to complex knowledge engineering problems requires methods from one or more of the following domains: machine learning, product design and development, software engineering, and experimental statistics. Without a structured process for handling the integration of these domains, the selection and application of methods is often ad-hoc, and, being ad-hoc, suffers from a number of problems including: inefficiencies due to applying the wrong method or tool for the problem, incomplete solutions that do not fully address the problem under consideration, and low-quality by emphasis on the technical aspects of the problem rather than users' needs. The combination of these factors, in particular the lack of attention to the user and organizational needs, frequently results in the development of solutions that do not yield the desired results.

In order to address these issues we need a framework to organize or structure the relevant subject-matter domains and the diverse set of methods and tools from these domains necessary for a comprehensive solution to the problem at hand. The development of this framework, addressed in Chapter 3, is the primary research problem and main contribution of this thesis.

3 Representation-Based Models for Knowledge Engineering

The purpose of this chapter is to develop the theoretical framework for the selection and integration of methods and tools from multiple engineering domains within the context of knowledge engineering problems. To this end, the chapter is organized into four parts. Section 3.1 develop our five layered representation-based model, called the Integrated Meta-Representational Model (IMRM), that allows complex knowledge engineering problems to be structured so that the necessary domains and associated methods and tools can be determined, and then applied toward the solution of the knowledge engineering problem. Section 3.2 provides a brief literature survey of existing work related to representation and problem-solving that we used in creating the IMRM. Section 3.3 is an overview for how to apply the IMRM to knowledge engineering problems. Lastly, Section 3.4 illustrates the process of applying the IMRM to the three knowledge engineering problems developed in Chapter 2 and discusses the resulting representation-based models.

3.1 The Integrated Meta-Representational Model

Real-world knowledge engineering problems are, in general, complex and ill-structured. The combination of multiple engineering domain and overlapping alternative methods inside each of these domains results in a large number of possible approaches to solving any particular knowledge engineering problem. Furthermore, different domains tend to conceptualize and represent knowledge differently which makes it difficult to integrate methods across domains.

In order to apply methods from different engineering domains we need a common representational language for problem-solving. To this end, we have developed a representation-based approach that provides an integrated layering of problem elements, engineering domains, and methods and tools from these domains in order to solve the problem of interest. This functional layering of a problem provides an abstract subject-matter domain-neutral model that then enables the selection of the appropriate subject matter domains, and, in concurrence the associated methods and tools for each layer of the representation. The Integrated Meta-Representational Model (IMRM) ([Desa and Munger, 2013], [Munger et al., 2015]), consisting of five sequential problem-solving layers shown in Figure 3.1, provides the theoretical framework of our representation-based approach.

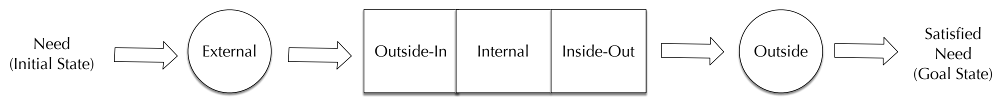


Figure 3.1: The Integrated Meta-Representational Model

The IMRM begins with an initial state, called the *External* layer, that represents how the problem under consideration is solved at the present time. From this initial state, the IMRM progressively works towards the goal state, called the *Outer* layer, which provides the solution to the problem. The five layers of the IMRM that take us from the initial state to the goal state are defined as follows:

1. **External**: represents the context for the initial state in terms of the way in which things are done at the present time.
2. **Outside-In**: represents the transformation of the *External* layer into a set of functional requirements for the form of the desired solution.

3. **Internal**: represents the abstract functional specification of the *Outside-In* layer in a manner that enables the exploration and selection of form (design) for the desired function.

4. **Inside-Out**: represents the conversion of the *Internal* layer into a realizable form. This layer includes all the relevant domain knowledge necessary to the development or embodiment of the design created in the *Internal* layer.

5. **Outer**: represents the implementation process that transforms the *Inside-Out* layer into the actual solution.

3.2 Literature Survey for Representation-Based Models

The Integrated Meta-Representational Model (IMRM) is a representation-based approach to solving complex multi-disciplinary problems. The notion of representation is also used in the following domains:

1. **Cognitive Science**: Representation is used to model how the human brain engages in information processing [Bermudez, 2014].

2. **Artificial Intelligence**: Representation is used to model different states in a problem space and the transition between states in order to solve the problem [Russell et al., 2003].

3. **Engineering Design**: Representation is used during the conceptual design process to establish the desired product functions, search for working solutions, and create the design or form for the product ([Alexander, 1964], [Pahl and Beitz, 1996]).

The five layers of the IMRM are based on the following three ideas from work in the aforementioned domains. The first idea is the notion that problem-solving is a representation-based process. Recent work in cognitive neuroscience has shown that the human-brain is a representational system, i.e. the human-brain solves problems by developing representations appropriate for solving the problem ([Metzinger, 2003], [Revonsuo, 2009]). Based on this important idea we structure the problem-solving process using representational layers. These layers provide a natural, intuitive way of guiding the problem-solver through the process of creating the representations appropriate for solving the problem.

The process of solving a complex problem can be modelled using three different types of representations: an initial state which represents the problem to be satisfied, goal state which represents the desired solution to the problem (e.g. satisfaction of the need), and a set of intermediate or internal states to go from the initial state to the goal state [Simon, 1973]. The second idea in the IMRM is the distinction between external and internal representations in problem-solving. One of the important conclusions from work in artificial intelligence [Russell et al., 2003], cognitive science [Bermudez, 2014], and engineering design ([Alexander, 1964], [Pahl and Beitz, 1996]) is that the internal representation used by the problem-solver has a significant impact on the quality and speed of problem-solving. In studies using the Towers of Hanoi problem, it is shown that different representations of the same problem could require up to sixteen times the amount of time to solve [Kotovsky et al., 1985]. The *Internal* layer of the IMRM addresses the internal representation by having the problem-solver explicitly consider what representation of the problem is necessary to enable the rapid development of a high-quality solution. It is also important to note that internal and external models are also used in control theory [Franklin et al., 1994].

The third idea is the notion of integration between the layers. The Integrated Information Theory (IIT) [Tononi, 2004], from cognitive neuroscience, explains consciousness as a function of the integrated information in a system. The theory asserts that human brain integrates information, and that by integrating information increases the overall information content of the system which leads to consciousness. Following from this notion, we have integrated the five layers of the IMRM in order to increase the overall information content of the representational model and, thereby, reduce the cognitive burden on the problem-solver who is applying the model.

3.3 Applying the Integrated Meta-Representational Model

The Integrated Meta-Representational Model (IMRM) provides an abstract framework for layering of problem elements, engineering domains, and methods and tools. We refer to the layering of a particular problem as a representation-based model for the problem under consideration. This representation-based model provides a step-by-step process for how to apply the necessary methods and tools in order to efficiently develop a high-quality solution to the problem under consideration.

The overall process for applying the IMRM to create a representation-based model for a particular problem is as follows:

1. **Determine the subject matters being represented**: For each level of representation determine the appropriate subject matter that needs to be represented. For example, if we are developing a software system, the subject matter at the *Inside-Out* level of representation is the software design that specifies how the artifact will be constructed.

2. **Identify the necessary methods and tools**: For each subject matter identify the appropriate set of domains and tools necessary for representation. For example, if our subject matter is the software design of a system, then we will need tools from the domain of Software Engineering in order to represent the software architecture, data structures, control logic, etc.

3. **Integrate the selected tools**: The selected tools need to be integrated at two

levels. At the first level the individual tools for each level of representation must be integrated so that they can be used together to resolve the issues for that particular layer. At the second level, the tools across the five levels of representation must be integrated so that the artifact is consistently represented throughout the development process in order to solve the problem of interest.

It is important to note that the mapping of the representational needs to the layers of the IMRM, and the selection of the appropriate methods and tools, involve trial and error.

The representation-based model, resulting from applying the IMRM to a particular problem, has the following features that facilitate the rapid development of high-quality solutions:

1. Each layer is a representation (or map) of the necessary steps in the process of designing and developing a high-value solution to solve a particular problem. The five layers of the IMRM are labelled as follows: *External*, *Outside-In*, *Internal*, *Inside-Out*, and *Outer*. The first layer, the *External*, represents the initial state of the problem under consideration; while the fifth layer, the *Outer*, represents the complete solution to the problem.

2. As one progresses through the five layers of representation, the level of abstraction first increases (*External* to *Internal*) and then decreases (*Internal* to *Outer*). By focusing resources (people, time, money) at the appropriate level of abstraction, the IMRM allows for a more comprehensive approach to ensuring and maximizing the satisfaction of the customer needs; resolving trade-offs between quality, cost, and time; incorporating user feedback into prototyping / manufacturing activities.

3. The sequential and functional layering of IMRM supports the concurrent selection of the appropriate methods and tools and their placement in the proper layer. This

enables the functional ("input-output") integration of the selected methods and tools, and thereby, facilitates a seamless transition between the layers.

3.4 Representation-based Models for Knowledge Engineering Problems in the Computer Networking Domain

In this section we demonstrate the application the Integrated Meta-Representational Model to the three knowledge engineering problems developed in Section 2. The treatment of each problem is separated into two parts. First, we discuss the layering of the problem elements, the corresponding engineering domains, and the necessary set of methods and tools from each engineering domain. We then show the functional integration of the selected methods and tools which results in the representation-based model for solving the particular problem under consideration.

3.4.1 Representation-Based Model for Product Quality Monitoring and Assessment

Table 3.1 shows the layers, representational subject matters, domains, and methods for the problem of product quality monitoring and assessment. In order to represent the product domain knowledge at the *External* layer we have selected two tools from the engineering design domain: the Functional Analysis System Technique (FAST) [Fox, 1993] for representing the function and form of a product and the Failure Modes and Effects Analysis (FMEA) [Fox, 1993] method for identifying product failure modes. From the data mining domain we have drawn upon three well-known machine learning algorithms—decision trees, naive bayes, and support vector machines [Hastie et al., 2009]—to learn the mapping function at the *Outside-In* for labeling each service request with a corresponding failure mode. At the *Internal* layer, we use double exponential smoothing [Chopra, 2007] to estimate the nominal component of each failure mode time-history. At the *Inside-Out* layer we use a Shewart Control Chart [Lawson and Erjavec, 2001] to monitor the deviation component of each failure mode. The *Outer* layer uses the same engineering design tools, FAST and FMEA, from the *External* layer in order to assess product quality and make recommendations for quality improvement.

Figure 3.2 shows the integration of the methods and tools described above. The resulting representation-based model is implemented by sequentially stepping through each layer as follows:

1. ***External* layer**. Model the function and form of the product of interest using a FAST diagram. For each major product component, perform a Failure Modes and Effects Analysis (FMEA) to identify potential failure modes. Select a subset of critical product failure modes from the FMEA to monitor and assess.

2. ***Outside-In* layer**. manually label a small subset of training data with the corresponding failure modes using the FAST and FMEA. Use the labeled data as input to machine learning algorithms in order to learn the classification function. Select the most accurate classifier and use it to label the complete customer service request data set.

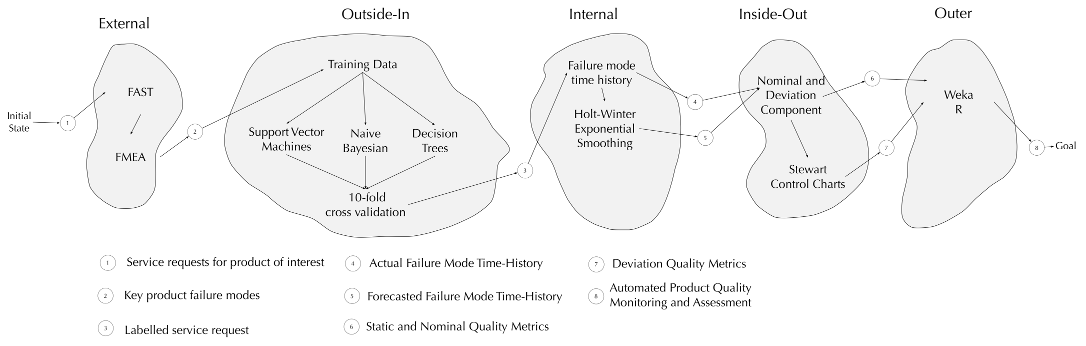
3. ***Internal* layer**: use the labelled service request data to create a time-history for each failure mode. The nominal component of each failure mode is estimated by the forecast from the Exponential Smoothing method. The deviation component of each failure mode is obtained by differencing the actual time-history and the nominal component estimate. (These concepts are defined in detail in Chapter 4)

4. ***Inside-Out* layer**: calculate the nominal metrics directly from the time-history of the nominal component. Calculate the deviation metrics by applying a Shewart control chart to the deviation component.

5. ***Outer* layer**: apply the assessment guidelines from the *External* layer to the static, nominal, and dynamic metrics computed at the *Inside-Out* layer.

Layer	Subject Matter	Domains and Representational Methods/Tools		
		Product Design and Development	Machine Learning	Time Series Analysis
External	Product Failure Modes	FAST and FMEA		
Outside-In	Machine Learning Model for Failure Mode Classification		Decision Trees, Naive Bayes, Support Vector Machines	
Internal	Time-Series Analysis			Double Exponential Smoothing
Inside-Out	Quality Monitoring			Control Charts
Outer	Quality Assessment	FAST and FMEA		

Table 3.1: Layer-Subject matter-representational tools/methods matrix for product quality monitoring and assessment



External
Outside-In
Internal
Inside-Out
Outer
Initial State
FAST
FMEA
Training Data
Support Vector Machines
Naive Bayesian
Decision Trees
10-fold cross validation
Failure mode time history
Holt-Winter Exponential Smoothing
Nominal and Deviation Component
Stewart Control Charts
Weka
R
Goal
1 Service requests for product of interest
2 Key product failure modes
3 Labelled service request
4 Actual Failure Mode Time-History
5 Forecasted Failure Mode Time-History
6 Static and Nominal Quality Metrics
7 Deviation Quality Metrics
8 Automated Product Quality Monitoring and Assessment

Figure 3.2: Representation-based model for product quality monitoring and assessment

The implementation the representation-based model, shown in Figure 3.2, is described in detail in Chapter 4.

3.4.2 Representation-Based Model for Predicting Customer Service Request Escalation

Table 3.2 shows the layers, representational subject matters, domains, and methods for the problem of predicting service request escalation. In order to capture and represent the technical, organizational, and financial considerations at the *External* layer

Layer	Subject Matter	Domains and Representational Methods/Tools		
		Knowledge Engineering	Machine Learning	Experimental Statistics
External	Organizational context and current work process for service request escalation	CommonKADS Organization, Agent, and Task Models		
Outside-In	Performance metric			Taguchi method: Signal-to-Noise Ratio
Internal	Experimental Design			Taguchi method: Orthogonal Array for Planned Experiments
Inside-Out	Perform Experiments		Weka Machine Learning Workbench	
Outer	Model Selection			Taguchi method: Analysis of the Means (ANOM)

Table 3.2: Layer-subject matter-representational tools/methods matrix for predicting service request escalation

we use three models from the CommonKADS methodology: the organization model, the agent, and the task model. The *Outside-In* layer uses signal-to-noise (s/n) ratios [Phadke, 1989], a technique from Robust Design, to define a quality characteristic and objective function for the escalation prediction model. At the *Internal* layer, we use the Taguchi methodology [Taguchi and Konishi, 1987] to create planned set of experiments for determining the optimal model with respect to the objective function. The *Inside-Out* layer we implement the experiments using the Weka machine learning work-bench [Witten and Frank, 2005]. Inside Weka we use three well-known classification learning algorithms: naive bayes, support vector machines, and random forests. At the *Outer* layer we analyze the results from the experiments and use the optimal parameters to generate a prediction model.

Figure 3.3 shows the representation-based model resulting from the integration of the methods and tools described above. The model is implemented by sequentially stepping through each layer as follows:

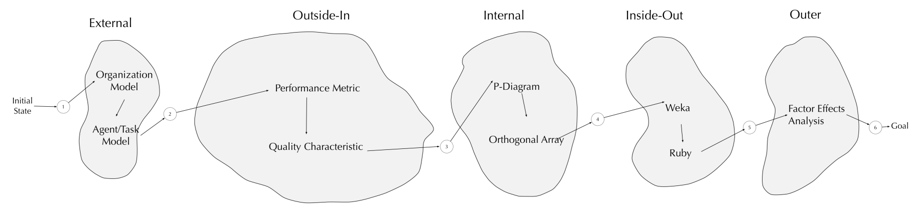


Figure 3.3: Representation-based model for predicting service request escalation

1. ***External* layer**: Capture the work-process that the machine learning model will automate by creating a CommonKADS Agent/Task Model.

2. ***Outside-In* layer**: Determine the performance metric that the machine learning model needs to optimize. Transform the performance metric into a signal-to-noise (s/n) ratio function to be maximized.

3. ***Internal* layer**: Determine the control, signal, and noise factors for the machine learning model. Select the appropriate orthogonal array and assign the factors and the factor settings or levels, respectively, to the columns and rows of the orthogonal array matrix.

4. ***Inside-Out* layer**: Perform the experiments in the Weka machine learning workbench. Record the prediction results for the test data-set and compute the s/n ratio for each experiment.

5. ***Outer* layer**: Perform Analysis of the Means (ANOM) to compute the s/n ratios for each combination of factor and level setting. Select the combination of factor settings that maximize the overall s/n ratio. Perform a verification experiment to check the results.

The implementation the representation-based model, shown in Figure 3.3, is described in detail in Chapter 5.

3.4.3 Representation-Based Model for the Development of Knowledge Engineering Software Products

Table 3.3 shows the layers, representational subject matters, domains, and methods for the development of knowledge engineering software products. The *External* layer uses the Organization and Agent/Task CommonKADS models [Schreiber, 1994] from the knowledge engineering domain to model the currently manual work process that needs to be automated. The *Outside-In* layer uses the House of Quality method [Hauser and Clausing, 1988] and UML Use Case diagrams [Schach, 2008] to capture the end-user needs and desired user experience for the KESP. The *Internal* layer uses the Function Structure [Pahl and Beitz, 1996], Morphological Matrix [Pahl and Beitz, 1996], and Utility Function [Pahl and Beitz, 1996] methods from the product design domain in order to explore different function realizations for the software automation and manage the inevitable trade-offs between quality and cost. The *Inside-Out* layer uses the Unified Modeling Language Component and Class diagrams [Schach, 2008] from the software engineering domain to create the software design for the KESP. The *Outer* layer uses the Iterative and Incremental development methodology [McConnell, 1996] from the software engineering domain to develop the KESP software implementation.

Figure 3.4 shows the information flow integrating the methods and tools from Table 3.3 into a representation-based model. Each information flow is depicted as directional arrow that shows the relationship between the inputs and outputs of two methods (with the exception of information flow (7) which involves the output of two different methods).

The representation-based model is implemented by sequentially stepping through each layer shown in Figure 3.4 starting with the initial state or high-level user need as follows:

1. ***External* layer**: take the high-level user need (see (1) in Figure 3.4) from the

initial state as input and create the CommonKADS Organization and Agent/Task models of the current work processes.

2. ***Outside-In* layer**: use the work process model (2) to create a House Quality and set of Use Case diagrams for the KESP. The *Outside-In* layer captures what the end-users want from the product and ensures that the KESP is high quality with respect to the user needs

3. ***Internal* layer**: use the user requirements (3)(5) and Use Case diagrams (4) to create the Function Structure, Morphological Matrix, and Utility Function for the KESP.

4. ***Inside-Out* layer**: create a UML Component diagram and corresponding set of UML Class diagrams to transform the design concept (6)(7) into a software design for the KESP.

5. ***Outer* layer**: implement the software architecture (8) and classes (9) to produce the KESP that satisfies the goal state (10).

The implementation the representation-based model, shown in Figure 3.4, is described in detail in Chapter 6.

Layer	Subject Matter	Domains and Representational Methods/Tools		
		Knowledge Engineering	Product Design	Software Engineering
External	Organizational context and currently manual Knowledge Engineering work process	CommonKADS Organization, Agent, and Task Models		
Outside-In	User needs for software automation		House of Quality	Use Case Diagrams
Internal	Conceptual design (functional specifications, solution-principles, design concept for the product)		Function Structure, Morphological Matrix, Utility Function	
Inside-Out	Software design (architecture, data structures, algorithms, control logic)			UML Component and Class Diagrams
Outer	Software development (planning, implementation, and testing)			Iterative and Incremental Development

Table 3.3: Layer-subject matter-Representational tools/methods matrix for knowledge engineering software products

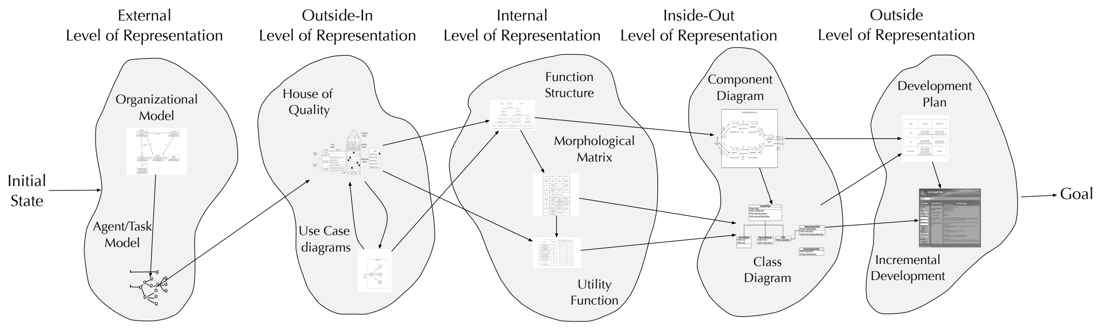


Figure 3.4: Representation-based model for knowledge engineering software product development

4 Structuring Technical Domain Knowledge: Theory and Application to Product Quality Monitoring and Assessment

In this chapter we address the application of the representation-based framework, developed in Chapter 3, to the knowledge engineering problem of monitoring and assessment of the quality of products in service. Since the data in this application is often unstructured and in the form of text exchanges between the customer and the enterprise, this problem involves the more general knowledge engineering research issue of modeling domain knowledge in order to produce meaningful results when mining unstructured data. Existing approaches share the common issue that the knowledge extraction results are often not properly structured for solving the engineering problem of interest and, therefore, require manual post-processing before they can be applied.

In this chapter we develop an approach to structuring technical domain knowledge that is based around the a priori modeling of the engineering problem of interest in order to enable: (1) efficient (rapid) collection, representation, and structuring of domain knowledge; and (2) the proper integration of domain knowledge with analytical KE methods in order facilitate the extraction of useful knowledge. We have demonstrate our approach to monitor and assess the quality of a network security product at a large computer networking company using a data set of 100,000 customer support cases

The chapter is organized as follows. Section 4.2 formulates the four distinct sub-problems involved in product quality monitoring and assessment. Section 4.3 assesses existing approaches to product quality monitoring and assessment with respect to the four sub-problems. Section 4.4 describes the application of the IMRM to the product quality monitoring and assessment problem. Section 4.5 shows the implementation of representation-based model using a real-world example involving customer service requests for a computer network security device. Section 4.6 provides the quality monitoring and assessment results for the computer network security device.

4.1 Introduction

Enterprises are increasingly attempting to transform the massive amounts of data being collected from customers into actionable knowledge that can be applied to design, development, and delivery of products and services. The purpose of this section is to properly motivate this problem, outline the research issues involved, and describe our key contributions.

4.1.1 Background

The engineering context for product quality monitoring and assessment can be organized into three cycles of activity: product design, development, and delvivery [Desa and Kannapan, 1995]. The product design, development, and delivery cycle (PD3) transforms a market need into a product delivered to that market. The product design and development cycle (PD2) transforms the market need into a product. The product design cycle (PD) transforms the market need into an embodiment design for the product [Pahl and Beitz, 1996].

Monitoring and assessment of product quality during PD2 (design and development) has been addressed by a large body of existing work. Several well-known approaches that have been used in industry are: control charts [Lawson and Erjavec, 2001], failure modes and effects analysis [Teng and Ho, 1996], the quality function deployment method [Hauser and Clausing, 1988], design of experiments [Phadke, 1989], and more recently six sigma [Harry and Schroeder, 2006]. However, relatively little work has addressed formal engineering methods for the monitoring and assessment of quality for products in service. The information that could be obtained from monitoring and assessing the quality of products in service has a wide range important applications to PD3 including: assessing the effectiveness of product design processes, improving the design of future products with respect to quality, and developing automated solutions to frequent customer problems.

The key challenge in quality monitoring and assessment during PD3 is obtaining suitable data for measuring product quality. Sensor networks have been used successfully used to monitor quality for large industrial products such as wind turbines [Garcia et al., 2006]. However, these sensors are expensive and monitoring them requires close cooperation with customers, which is not feasible for many applications. In this work we demonstrate how unstructured (or free-form text) problem descriptions from customer service requests, a type of data being collected by most enterprises that provide technical support for their products, can be used to monitor and assess the quality of products in service across a large number of customers.

4.1.2 Research Issues

The use of unstructured customer service requests for quality monitoring and assessment involves three general knowledge engineering research issues:

1. The structuring of subject-matter domain knowledge to organize unstructured data in complex technical domains.
2. The selection and integration of methods and tools from multiple domains.
3. The development of software environments for automating the application of the selected methods and tools for large data sets.

4.1.3 Contributions

Our contributions to the aforementioned research issues directly follow from the application of the Integrated Meta-Representational Model to the product quality monitoring and assessment problem, and are as follows:

1. We have applied the Integrated Meta-Representational Model to first identify, and then select methods and tools from three engineering subject matter domains—engineering design, machine learning, and time-series analysis—for addressing the quality monitoring and assessment problem (see Section 4.4).

2. We developed a process methodology, called the Product Quality Monitoring and Assessment Process Methodology (PQMAPM), that provides the theoretical framework for using the selected methods within the context of product quality monitoring and assessment. The key idea in the process methodology to translate the free-form text provided by the customer into engineering failure modes. To this end we use two methods from engineering design, the Function Analysis System Technique (FAST) and Failure Modes and Effects Analysis (FMEA), to collect and structure of the failure modes for the product of interest. We then apply a machine learning model, created using the support vector machines algorithm, to label each service request with a corresponding failure mode. Lastly, time-series analysis methods—exponential smoothing and control charts—are used to compute a number of metrics for monitoring product quality from the aggregated daily occurrence for each failure mode (see Section 4.5).

3. We have demonstrated the Product Quality Monitoring and Assessment Process Methodology within the real-world context of network security device. The quality metrics, computed using 100,000 customer service requests, have numerous practical product design, development, and delivery applications including: identifying product features with poor quality, improving service center resource allocation,

and providing an early warning system for potential quality problems (see Section 4.6).

4.2 Problem Formulation

The purpose of this section is to describe the problem of quality monitoring and assessment for products in service. To this end, we start by defining quality monitoring and assessment as it is used in this chapter. From this definition, we then organize quality monitoring and assessment into four distinct sub-problems using a real-world example involving a product manufacturer in the computer networking industry.

In our context, a high-quality product is one for which the frequency of occurrence of the failure modes is very low. Essentially a quality product is a reliable as measured by the failure rate. Monitoring and assessment are two distinct but related activities that are defined as follows [Phadke, 1989]:

1. **Quality Monitoring**: The ongoing measurement of the quality of a product as defined by a set of performance metrics
2. **Quality Assessment**: The use of the information obtained from quality monitoring to guide decisions related to improving product quality.

Monitoring and assessment can be performed during one or all three PD3 cycles. This work specifically addresses quality monitoring and assessment of products in service (the product delivery cycle of PD3). In order to concretely draw out the distinct sub-problems involved in this context consider the following quality monitoring and assessment problem involving a network security product.

The Cisco ASA 5505 is a network security product that provides a combination firewall, intrusion detection, and virtual private network client/server for small to medium businesses. In addition to the standard network security related functionality, an important function of the ASA 5505 is the so-called fail-over feature that allows two separate ASA 5505 products to be operating simultaneously in order to provide redundancy in the case of a failure.

In order to enable the fail-over feature, a customer will designate one ASA 5505 as the primary network device. This device is then powered on and configured normally to process network traffic. The second device is then powered on, but is configured as a secondary device that does not process network traffic. Once both devices are powered on, the primary device will periodically send the secondary device an update message containing the current state of the network. When the secondary device does not receive an update from the primary device after a certain specified period of time it will take over the responsibilities of the primary device and start processing network traffic based on the most recent update it has received.

The product manufacturer would like to monitor the failure modes for products in service in order to determine and improve the quality level of the ASA 5505 fail-over feature. Some examples of the monitoring and assessment issues related to improving the quality of the ASA 5505 fail-over feature are as follows:

1. Determine the ASA 5505 sub-systems which have relatively high failure mode rates.
2. Forecast the expected occurrence of fail-over related failure modes for the ASA 5505.
3. Determine the failure modes, if any, that are increasing in frequency of occurrence over time.
4. Identify new, previously unknown, fail-over related failure modes.

In order to monitor and assess the quality of the ASA 5505, the manufacturer has collected a data set of service requests that were submitted by customers to the manufacturer's technical support center. Each service request consists of an unstructured (free-form text) customer description of a failure mode that occurred and a time-stamp. The problem description poses the following knowledge engineering challenges: customers can use different language to describe the same failure mode; the failure mode is described using computer-network domain-specific terminology; duplicate or irrelevant information. An actual customer problem description of an ASA 5505 fail-over issue which illustrates these challenges is shown in Figure 4.1.

> We have one ASA with 2 ISP connections and several site to site VPNs set-up to other sites with ASAs. When we test failing over the VPNs to use the second ISP connection the tunnels are not coming up. We have confirmed that outbound routing fails over immediately.

Figure 4.1: Example service request for the ASA 5505

Monitoring and assessing product quality using the service request data involves the following four sub-problems:

1. **Failure Modes Definition**: Determine the set of failure modes that we would like to monitor for the product of interest
2. **Failure Mode Quantification**: Measure the daily occurrence of each failure mode in the collected service requests. (The term quantification comes from related work [Forman et al., 2006] that addresses quantifying the occurrence of problems in customer service centers.)
3. **Time-series Analysis**: Determine the nominal (expected) daily occurrence of each failure mode and the deviation of the actual daily occurrence from its nominal value.
4. **Quality Metrics Computation**: Define and compute a set of metrics for char-

acterizing the occurrence of each failure mode.

The four sub-problems can be described more formally as follows:

First, we need to work with experts to identify the failure modes that we would like to monitor. Let m denote the total number of failure modes to be monitored and c_j be the jth $(j = 1, 2, ..., m)$ failure mode for the product of interest. This is sub-problem 1, failure mode definition, described above.

We must then quantify the occurrence of each identified failure mode c_j $(j = 1, 2, ..., m)$ based on the received customer service requests. Let n be the total number of service requests received for the particular product we would like to monitor and d_i be the ith $(i = 1, 2, ...n)$ service request received. If we assume that each service request is reporting one product failure mode then we need to map each service request $d_i \in D$ to a corresponding failure mode $c \in C$ as output. The failure mode quantification problem is to create the a mapping function, denoted Φ, and apply it in order to label each service request d_i $(i = 1, 2, ..., n)$ with the corresponding failure mode $c \in C$. This is sub-problem 2, quantification, described above.

Once we have the time-history of each failure mode, the third sub-problem is to compute the metrics for monitoring product quality from time-history of each failure mode c_j $(j = 1, 2, ..., m)$. The specific metrics for monitoring the product failure modes will vary depending on the problem under consideration and the product of interest.

The three categories or types of metrics that are generally useful for monitoring product quality are as follows. First, we can monitor the actual quality of the products in service, e.g. how many service requests were received for a particular product failure mode. Second, we can measure the expected, or nominal, product quality for the product of interest, e.g. how many service requests do we expect to receive for a particular product failure mode today. Third, we can measure the deviation or the difference between actual product quality and the expected product quality.

If $y_t^{c_j}$ denotes the number of service requests with failure mode c_j in day t, and l is the number of days we are monitoring then the time-history of c_j is

$$Y^{c_j} = \{y_1^{c_j}, y_2^{c_j}, ..., y_l^{c_j}\} \tag{4.1}$$

The calculation of the nominal and deviation quality metrics requires the nominal and deviation components associated with each failure mode time-history. We assume that the nominal occurrence of each failure mode can be characterized using three components: level, trend, and seasonality. The level, $L_t^{c_j}$ characterizes the failure mode's mean daily occurrence, trend, $T_t^{c_j}$ characterizes the rate of change in this mean occurrence over time, and the seasonality, $S_t^{c_j}$ characterizes recurring cyclic patterns. If $\bar{y}_t^{c_j}$ is the nominal occurrence for failure mode c_j on day t, then

$$\bar{y}_t^{c_j} = [L_t^{c_j} + T_t^{c_j}] \times S_t^{c_j} \tag{4.2}$$

where $L_t^{c_j}$, $T_t^{c_j}$, and $S_t^{c_j}$ are the level, trend, and cyclicity of the failure mode c_j on day t. The time-history of the nominal component, $\bar{Y}^{c_j}$, is then defined as follows:

$$\bar{Y}^{c_j} = \{\bar{y}_1^{c_j}, \bar{y}_2^{c_j}, ..., \bar{y}_l^{c_j}\} \tag{4.3}$$

The deviation,$\tilde{y}_t^{c_j}$, between actual occurrence of failure mode c_j and the nominal component $\bar{y}_t^{c_j}$ at time t. If $\tilde{y}_t^{c_j}$ is the deviation component at time t, then

$$\tilde{y}_t^{c_j} = y_t^{c_j} - \bar{y}_t^{c_j}, (j = 1, 2, ..., m). \tag{4.4}$$

and the time-series history of the deviation component is

$$\tilde{Y}^{c_j} = \{\tilde{y}_1^{c_j}, \tilde{y}_2^{c_j}, ..., \tilde{y}_l^{c_j}\}, (j = 1, 2, ..., m). \tag{4.5}$$

Procedural, the monitoring and assessment problem becomes: First, construct the time-history Y^{c_j} for each failure mode c_j $(j = 1, 2, ..., m)$. Second, estimate the time-history of the nominal component $\bar{Y}^{c_j}$. Three, determine the deviation component $\tilde{Y}^{c_j}$ for each failure mode c_j $(j = 1, 2, ..., m)$. Four, compute the product quality metrics from the actual, nominal, and deviation time-histories for each failure mode.

4.3 Literature Survey

The purpose of this section is to survey existing work related to product quality monitoring and assessment and motivate the need for a new multi-disciplinary approach. To this end, the section is organized into three parts as follows. First, we discuss several well-known approaches to quality monitoring and assessment that are currently used during product design, development, and delivery. Next, we discuss other work that addresses one or more of the four sub-problems defined in Section 4.2, but does not explicitly address product quality monitoring and assessment. Lastly, we assess the applicability of the related work to the problem under consideration: quality monitoring and assessment using unstructured service request data.

4.3.1 Existing Approaches to Product Quality Monitoring and Assessment

Existing approaches for product quality monitoring and assessment can be organized into three different groups corresponding to where they fit in the product design, development, and delivery process. During product design the objective of quality monitoring and assessment is to minimize defects with the product's design. Failure Modes and

Effects Analysis (FMEA) ([Fox, 1993], [Teng and Ho, 1996]) is a well-known and widely used technique for identifying different product failure modes and assessing the risk associated with these failure modes. The FMEA process involves breaking the product down into smaller functional sub-systems, identifying potential failure modes of these sub-systems and their causes, determining the necessary actions that need to be taken in order to resolve the failure mode, and finally assessing the risk associated with each failure mode. The risk of each failure mode is represented by a risk priority number (RPN) which is defined as the product of three quantities: severity of the failure mode (S), difficulty of detection (D), and frequency of occurrence (O).

In addition to the FMEA there are several other well-known techniques for failure mode and risk analysis. Fault tree analysis (FTA) [Lee et al., 1985] maps out the set of events leading up to a failure mode's occurrence using a tree constructed from boolean logic gates. The resulting tree can be used to reason about the root cause of a failure mode as well as monitor the system or product to detect potential failure modes before they happen. The Structured What-If Technique (SWIFT) [Card et al., 2012] is a systems-based risk identification technique that focuses on high-level processes and can often be conducted more quickly than an FMEA or FTA. The SWIFT technique uses structured brainstorming in combination with pre-defined prompts to examine risks of different system failure modes.

Once the product design is finalized and the product is ready for development and manufacturing, the objective of quality monitoring and assessment is to minimize variation in the manufacturing processes. Six Sigma [Harry and Schroeder, 2006] is a well-known approach to minimizing process variation consisting of five distinct phases: define, measure, analyze, improve, and control. The Six Sigma process provides a set of qualitative tools for identifying the different process variables that need to be monitored, and a set of statistical tools for monitoring these variables in order to detect variations in quality.

4.3.2 Related Work in Machine Learning and Time-Series Analysis

In the Section 4.2 we defined four distinct sub-problems—domain knowledge representation, quantification, monitoring, and assessment—related to product quality monitoring and assessment. Related work that addresses these problems falls into two high-level groups. The first group is work in data mining and machine learning that addresses the domain knowledge representation and quantification problems. The second group is the work in time-series analysis that addresses the monitoring and assessment sub-problems.

Within our area of interest, unstructured customer service data, there are two general approaches to domain knowledge modeling and quantification. The first approach, is to use machine learning algorithms to automatically generate a domain knowledge model from the data. Generally this involves text clustering, such as TroubleMiner [Medem et al., 2009] which uses hierarchical clustering to automatically categorize network support cases based on the text in the customer problem descriptions. These clusters are then reviewed by experts to identify interesting patterns. Text clustering has the advantage of requiring relatively little human effort up-front in the data mining process. However, this benefit is offset by its sensitivity to small linguistic differences in how a customer describes a problem, which can cause unrelated support cases to be grouped together, and similar support cases to be split across multiple clusters.

The second approach to quantification involves combining machine learning algorithms with inputs from human experts. Several examples of this approach in the context of customer service data are as follows. The Incident Categorization and Analysis system [Forman et al., 2006] addresses the quantification of frequent issues in customer support cases using a process that begins with a k-means clustering to create an initial candidate set of issues. These issues are then manually refined by experts, and then used as the prediction labels for categorizing incoming support cases. The Unstructured Information Modeller system [Spangler and Kreulen, 2007] also addresses the quantification of frequent help desk issues but starts by having experts define important terms and phrases. These frequent terms and phrases are used as attributes for creating an initial clustering of the support cases. Human experts then refine this initial clustering through deleting, merging, and splitting clusters. The final clustering is then automatically applied to incoming data. The Netsieve system [Potharaju et al., 2013] addresses the problem of tagging problem and resolution pairs in network support cases. Frequent phrases in support cases are first manually mapped by human experts into a ontology that relates customer problems and the actions taken by the support engineers to resolve the problem. This model is then applied to tag problems and solutions in new support cases. Although experts can decipher complex technical vocabulary in order to merge similar problems and separate clusters with dissimilar problems, this approach is still largely data driven. The initial domain knowledge model is determined by the structure of the data and not the underlying subject-matter domain. Consequently, the results generally require a significant amount of post-processing before they can be used to solve engineering problems.

Within the time-series analysis domain we will focus on a subset related work in the area of Biosurveillance that involves the monitoring and assessment of medical data in order to detect disease outbreaks and, consequently, involves many of the same issues as the monitoring and assessment sub-problems discussed in Section 4.2. Biosurveillance monitoring methods typically consist of two parts: a forecasting component which is used to create a nominal (expected) model for the data, and a monitoring component that raises an alert when the actual behavior significantly deviates from the nominal model. A number of different approaches have been explored for implementing the forecasting and monitoring components. The Early Aberation and Reporting System (EARS) and BioSense [Lotze and Shmulei, 2008] systems developed by the Center for Disease Control (CDC) use a moving average forecasting component combined with a control chart monitoring component to monitor hospital records for disease outbreaks and bio-terrorism. The ESSENCE system [Lombardo et al., 2004] developed by the Department of Defense (DOD) uses regression to forecast the next day‘s value and then monitors the residuals (forecast error) using a control chart.

Google Flu Trends [Ginsberg et al., 2009] provides an approach to monitoring unstructured data to predict influenza trends. The number of Google search queries related to influenza is quantified using a linear model that relates the number of influenza related physicians visits to the number of influenza related Google search queries. The parameters to this model are learned by finding the combination of Google search queries that fit historical Center for Disease Control (CDC) data for influenza historical CDC data. The model is then applied to incoming Google search queries to provide a real-time estimate of influenza cases.

4.3.3 Assessment of Related Work

Quality monitoring and assessment using unstructured customer data involves four distinct sub-problems: domain knowledge representation, quantification, monitoring, and assessment (Section 4.2). The related work discussed above provides partial solutions to each of these sub-problems, however, no single piece of related work addresses all four sub-problems. In order to provide a comprehensive solution to the four quality monitoring and assessment sub-problems we need an integrated multi-disciplinary approach that uses methods from engineering design, data mining, and time-series analysis.

Engineering design methods, in particular Failure Modes and Effects Analysis, provide a structured process for capturing and representing domain knowledge about a product in a format that enables quality assessment. However, these tools are generally applied manually and not directly suitable to the unstructured service request data involved in product quality monitoring and assessment. The work [Garcia et al., 2006] that does address quality monitoring during product support involves using sensors to periodically measure product performance. The need for sensors to be installed on the product in service limits the application of these methods to highly-specialized low-volume products, such as wind turbines, and are therefore not generally suitable to monitoring large number of products in the field.

While data mining provides machine learning techniques, such as K-Nearest Neighbors clustering, that facilitate the automatic generation of a domain knowledge representation from data with a minimal up-front investment of human time and effort, the complex technical vocabulary and domain knowledge in engineering application domains—such as computer networks—makes it difficult to generate a useful domain knowledge representation through clustering or frequent phrase extraction. Consequently, the results generated by these algorithms frequently require a significant amount of post-processing before they can be used to address product design, development, and delivery problems. The time-series analysis domain provides statistical methods for monitoring time-series data. However, the application of these methods to monitor unstructured data generally is not generally feasible due to massive computational requirements and the need for large amounts of training data.

4.4 Theory: Representation-based Model for Product Quality Monitoring and Assessment

In order to structure and resolve the four sub-problems involved in quality monitoring and assessment we apply the Integrated Meta-Representational Model developed in Chapter 3. The use of representation enables these four sub-problems to be cleanly separated into distinct layers where the appropriate engineering domains and associated methods for resolving these sub-problems can be identified and applied. The *External* and *Outer* layers of the IMRM represent the initial and goal state for problem under consideration. The domain knowledge representation sub-problem, where we capture the domain knowledge about the product, maps to the *External* layer. Likewise, the assessment sub-problem maps to the *Outer* layer because it satisfies the goal of product quality monitoring and assessment. For the product quality monitoring and assessment problem we need to get from the product quality metrics at the *External* layer to the product quality assessment at the *Outer* layer. The abstract representation of the service request data that enables quality monitoring and assessment is the nominal and deviation components of each failure mode's time-history. These components are necessary to compute the static, nominal, and deviation metrics for monitoring product quality. The quantification sub-problem maps to the *Outside-In* layer where we go from the product failure modes at the *External* layer to the failure mode time-history at the *Internal* layer. Similarly, the monitoring problem maps to the *Inside-Out* layer which takes us from the failure mode time-histories at the *Internal* layer to the assessment at the *Outer* layer.

Table 4.1 shows the layers, representational subject matters, domains, and methods for the problem of product quality monitoring and assessment. In order to represent the product domain knowledge at the *External* layer we have selected two tools from the engineering design domain: the Functional Analysis System Technique (FAST) [Fox, 1993] for representing the function and form of a product and the Failure Modes and Effects Analysis (FMEA) [Fox, 1993] method for identifying product failure modes. From the data mining domain we have drawn upon three well-known machine learning algorithms—decision trees, naive bayes, and support vector machines [Hastie et al., 2009]—to learn the mapping function at the *Outside-In* for labeling each service request with a corresponding failure mode. At the *Internal* layer, we use double exponential smoothing [Chopra, 2007] to estimate the nominal component of each failure mode time-history. At the *Inside-Out* layer we use a Shewart Control Chart [Lawson and Erjavec, 2001] to monitor the deviation component of each failure mode. The *Outer* layer uses the same engineering design tools, FAST and FMEA, from the *External* layer in order to assess product quality and make recommendations for quality improvement.

Layer	Subject Matter	Domains and Representational Methods/Tools		
		Product Design and Development	Machine Learning	Time Series Analysis
External	Product Failure Modes	FAST and FMEA		
Outside-In	Machine Learning Model for Failure Mode Classification		Decision Trees, Naive Bayes, Support Vector Machines	
Internal	Time-Series Analysis			Double Exponential Smoothing
Inside-Out	Quality Monitoring			Control Charts
Outer	Quality Assessment	FAST and FMEA		

Table 4.1: Layer-Subject matter-representational tools/methods matrix for product quality monitoring and assessment

Figure 4.2 shows the representation-based model resulting from the integration of the methods and tools described above. The model is implemented by sequentially stepping through each layer shown in Figure 4.2 as follows:

1. ***External* layer**: model the function and form of the product of interest using a FAST diagram. For each major product component, perform a Failure Modes and Effects Analysis (FMEA) to identify potential failure modes. Select a subset of critical product failure modes from the FMEA to monitor and assess.

2. ***Outside-In* layer**: manually label a small subset of training data with the corresponding failure modes using the FAST and FMEA. Use the labeled data as input to the naive bayes, decision trees, and Support Vector Machine learning algorithms in order to learn the classification function. Select the most accurate classifier and use it to label the complete customer service request data set.

3. ***Internal* layer**: use the labelled service request data to create a time-history for each failure mode. The nominal component of each failure mode is estimated by the forecast from the Exponential Smoothing method. The deviation component of each failure mode is obtained by differencing the actual time-history and the

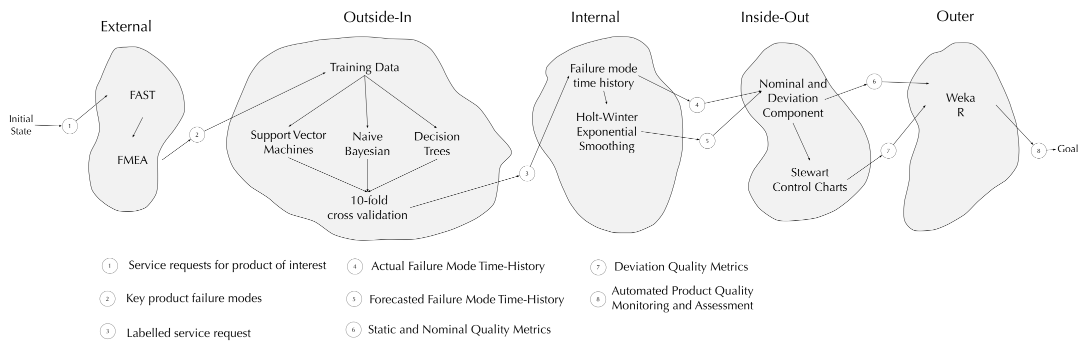

External
Outside-In
Internal
Inside-Out
Outer
Initial State
FAST
FMEA
Training Data
Support Vector Machines
Naive Bayesian
Decision Trees
10-fold cross validation
Failure mode time history
Holt-Winter Exponential Smoothing
Nominal and Deviation Component
Stewart Control Charts
Weka
R
Goal
1 Service requests for product of interest
2 Key product failure modes
3 Labelled service request
4 Actual Failure Mode Time-History
5 Forecasted Failure Mode Time-History
6 Static and Nominal Quality Metrics
7 Deviation Quality Metrics
8 Automated Product Quality Monitoring and Assessment

Figure 4.2: Representation-based model for product quality monitoring and assessment

nominal component estimate.

4. ***Inside-Out* layer**: calculate the nominal metrics directly from the time-history of the nominal component. Calculate the deviation metrics by applying a Shewart control chart to the deviation component.

5. ***Outer* layer**: apply the assessment guidelines from the *External* layer to the static, nominal, and dynamic metrics computed at the *Inside-Out* layer.

The comprehensive (step-by-step) implementation of the representation-based model is provided in the following section (Section 4.5).

4.5 Application: Process Methodology for Product Quality Monitoring and Assessment

In this section we describe the process methodology for implementing the representation-based model for product quality monitoring and assessment developed in the previous section. The process methodology is organized into five parts—*External*, *Outside-In*, *Internal*, *Inside-Out*, and *Outer*–corresponding to five layers of the Integrated Meta-Representational Model in Figure 4.2. For each layer we first identify the overall inputs and outputs and the layer's role in creating a solution to the overall problem. We then describe the methods and techniques used at the layer to transform the inputs into outputs. Lastly we present the implementation of each layer for the product quality monitoring and assessment problem of interest. The methods and techniques at each layer are illustrated using the quality monitoring and assessment problem described in Section 4.2 for the ASA 5505 computer network security product.

4.5.1 External Layer: Domain Knowledge Modeling

The problem addressed at the *External* layer is collecting and representing the domain knowledge about the product for which we would like to monitor and assess. This involves modeling both how the product works as well as the different product failure modes. To this end, we use the Functional Analysis System Technique (FAST) [Fox, 1993] method to represent the relationships between product functionality and product sub-systems. The Failure Modes and Effects Analysis (FMEA) [Fox, 1993] is then used to collect and organize the failure modes associated with each product sub-system.

The process for applying the FAST and FMEA methods to model the domain knowledge for the product is as follows:

1. Determine the function and form relationships for the selected product using the **Functional Analysis System Technique** (FAST) [Fox, 1993]. The FAST diagram is created as follows: a) Write the primary function of the product on the right side of the diagram; b) Break the product down into logical elements or sub-systems. Write these sub-systems on the left side of the diagram; c) Work from both ends of the diagram, by adding the relevant product sub-functions, until the primary function and sub-systems are connected.
2. Perform a **Failure Modes and Effects Analysis** (FMEA) [Fox, 1993] for each product sub-system identified in the FAST diagram from Step 1. The FMEA process is as follows: a) Work with product experts—typically these experts would

be selected from the product design, development, and delivery team(s) for the product of interest—to determine the potential failure modes for each product sub-system identified in the FAST diagram; b) Describe the symptoms (effects) associated with each failure mode; c) Characterize the risk associated with each failure mode using a Risk Priority Number (RPN). The RPN is calculated as the product of the severity (S), frequency of occurrence (O), and difficulty of detection (D) rated on an 8 point scale; d) Determine the appropriate actions to take in order to resolve the failure mode; e) Organize the results into an FMEA table as shown in Figure 4.4.

3. Determine the set of failure modes to use for monitoring and assessing product quality. The process for selecting the failure modes to monitor is as follows: a) Select a value for m, the total number of failure modes that we would like to monitor; b) Use the RPN values from the FMEA to rank the failure modes; c) Select the top m failure modes corresponding to the m largest RPN values.

4. Define the metrics for monitoring the selected failure modes. While the specific metrics can vary depending on the product of interest, there are three general types of metrics for monitoring product quality. First, we want to measure the overall quality of the product, e.g. the total number of times a failure mode occurs. We refer to this type of quality as static because it is not a function of time. Second, we want to measure the nominal or expected occurrence for a particular failure mode over time. Third, we want to measure how the actual occurrence is deviating from this nominal occurrence.

We now illustrate the implementation of the *External* layer for our problem of monitoring and assessing product quality for the ASA 5505 network security device. Figure 4.3 shows the FAST diagram for the fail-over feature of the ASA 5505. Fail-over is implemented by three sub-systems: “fail-over link”, “stateless fail-over”, and “stateful

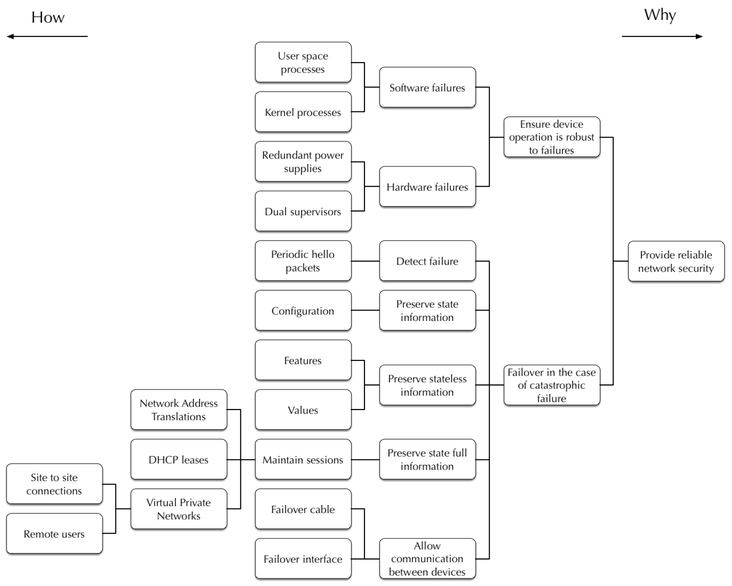
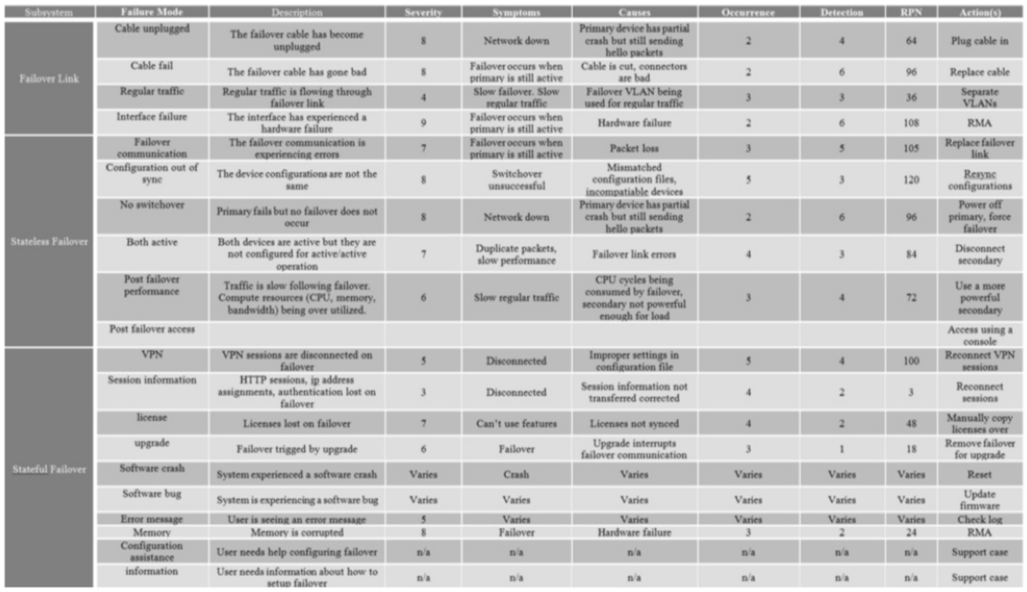


Figure 4.3: FAST diagram for ASA5505 fail-over sub-system

fail-over". The stateless fail-over sub-system realizes the functionality related to maintaining an identical device configuration across the two ASA 5505 products and ensures that the backup will operate identically to the primary when/if fail-over occurs. The stateful fail-over sub-system realizes the functionality related to preserving the state, e.g. connections and IP addresses, of the computer network when the fail-over occurs and makes the fail-over process transparent to the end-users on the network. Finally, the fail-over link sub-system realizes the functionality related to enabling communication between the primary and backup ASA 5505 and triggers fail-over in the case that the primary ASA 5505 experiences a failure.

The Failure Modes and Effects Analysis (FMEA) identified 20 key failure modes for the three ASA 5505 sub-systems related to fail-over. For each of the 20 failure modes, shown in Figure 4.4, we worked with experts to determine the values for S, D, and O in order to calculate a Risk Priority Number (RPN). We then selected the 10 failure modes with the highest RPN values as the failure modes to monitor. Table 4.2 contains



Subsystem	Failure Mode	Description	Severity	Symptoms	Causes	Occurrence	Detection	RPN	Action(s)
Failover Link	Cable unplugged	The failover cable has become unplugged	8	Network down	Primary device has partial crash but still sending hello packets	2	4	64	Plug cable in
	Cable fail	The failover cable has gone bad	8	Failover occurs when primary is still active	Cable is cut, connectors are bad	2	6	96	Replace cable
	Regular traffic	Regular traffic is flowing through failover link	4	Slow failover. Slow regular traffic	Failover VLAN being used for regular traffic	3	3	36	Separate VLANs
	Interface failure	The interface has experienced a hardware failure	9	Failover occurs when primary is still active	Hardware failure	2	6	108	RMA
Stateless Failover	Failover communication	The failover communication is experiencing errors	7	Failover occurs when primary is still active	Packet loss	3	5	105	Replace failover link
	Configuration out of sync	The device configurations are not the same	8	Switchover unsuccessful	Mismatched configuration files, incompatiable devices	5	3	120	Resync configurations
	No switchover	Primary fails but no failover does not occur	8	Network down	Primary device has partial crash but still sending hello packets	2	6	96	Power off primary, force failover
	Both active	Both devices are active but they are not configured for active/active operation	7	Duplicate packets, slow performance	Failover link errors	4	3	84	Disconnect secondary
	Post failover performance	Traffic is slow following failover. Compute resources (CPU, memory, bandwidth) being over utilized.	6	Slow regular traffic	CPU cycles being consumed by failover, secondary not powerful enough for load	3	4	72	Use a more powerful secondary
	Post failover access								Access using a console
Stateful Failover	VPN	VPN sessions are disconnected on failover	5	Disconnected	Improper settings in configuration file	5	4	100	Reconnect VPN sessions
	Session information	HTTP sessions, ip address assignments, authentication lost on failover	3	Disconnected	Session information not transferred corrected	4	2	3	Reconnect sessions
	license	Licenses lost on failover	7	Can't use features	Licenses not synced	4	2	48	Manually copy licenses over
	upgrade	Failover trigged by upgrade	6	Failover	Upgrade interrupts failover communication	3	1	18	Remove failover for upgrade
	Software crash	System experienced a software crash	Varies	Crash	Varies	Varies	Varies	Varies	Reset
	Software bug	System is experiencing a software bug	Varies	Varies	Varies	Varies	Varies	Varies	Update firmware
	Error message	User is seeing an error message	5	Varies	Varies	Varies	Varies	Varies	Check log
	Memory	Memory is corrupted	8	Failover	Hardware failure	3	2	24	RMA
	Configuration assistance	User needs help configuring failover	n/a	n/a	n/a	n/a	n/a	n/a	Support case
	information	User needs information about how to setup failover	n/a	n/a	n/a	n/a	n/a	n/a	Support case

Figure 4.4: FMEA for ASA 5505 fail-over sub-system

a brief description of the selected failure modes.

For this problem we define two static metrics, absolute and relative occurrence, for measuring how often a particular failure mode has occurred across all service requests received. In order to measure the nominal occurrence or expected daily occurrence of each failure mode we use the mean occurrence (failure modes/day) and the growth or rate of change in this mean occurrence over time (failure modes/day).

The deviation metrics, measure the deviation of the actual daily occurrence from the nominal (expected) daily occurrence. We use two metrics for measuring when this deviation is significantly large. The absolute deviation metric measures the total number of days when a failure mode's occurrence significantly deviates from the nominal. The relative deviation metric measures number of days that the deviation component is larger than the acceptable user-defined threshold relative to the total number of days

Failure Mode	Description
Both active	Fail-over occurs when the primary device is still active
Configuration sync	The software configuration of the secondary device does not match the primary device
No Switch-over	The fail-over does not occur when the primary device fails
Fail-over communication	Normal traffic is being routed over the fail-over link
Fail-over link	The fail-over cable connecting the primary and secondary devices is experiencing an error
Hardware	The hardware fail-over controller is not operating correctly
Software upgrade	The user needs to upgrade the software version on the secondary device
License	The software license of the secondary device does not match the primary device
Memory	Contents of the primary device‘s main memory was corrupted during failover
VPN	VPN sessions were disconnected on fail-over

Table 4.2: Failure modes for the ASA 5505 fail-over feature

monitored.

4.5.2 Outside-In Layer: Machine Learning

The *Outside-In* layer addresses the problem of labeling each service request with a corresponding failure mode from the set of failure modes identified at the *External* layer for the ASA 5505 fail-over feature the failure mode labels are shown in Table 4.2. The failure mode labels are then used at the Internal layer to aggregate the service requests by failure mode and create the time-series of the occurrence of each failure mode.

Let d_i $(i = 1, 2, ..., n)$ be the set of service requests collected over a time-horizon of l days. Each service request d_i is a tuple consisting of two values:

$$d_i \triangleq (\pi_i, \tau_i). \tag{4.6}$$

where π_i is a text description of the failure mode that is being experienced by the customer and τ_i is the time-stamp indicating the day t $(t = 1, 2, ..., n)$ that d_i was received.

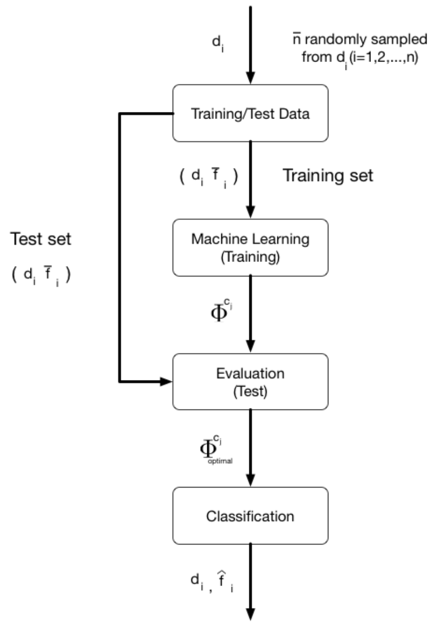
For each service request d_i we must determine the appropriate failure mode label from the set of failure modes c_j $(j = 1, 2, ..., m)$. We refer to the failure mode label for d_i as f_i where the value of f_i can be any one of the m failure modes that we are monitoring:

$$f_i \in \{c_1, c_2, ..., c_m\}. \tag{4.7}$$

In order to determine f_i $(i = 1, 2, ...n)$ we create a classification function that maps each service request d_i $(i = 1, 2, ..., n)$ to a corresponding failure mode c_j $(j = 1, 2, ...m)$:

$$\Phi : d_i \rightarrow \{c_1, c_2, ..., c_m\}. \tag{4.8}$$

The classification function, Φ, is created using a machine learning process shown in Figure 4.5. First, we create a training/test data-set of service requests by manually determine the failure mode label for a small sample of service requests. This labeled data is then split into separate training and test sets. The training set is used as input to the machine learning algorithms for learning the classification function. The test set is used to evaluate the performance of the resulting classification functions. The optimal classification function is then selected and used to label the complete service request data-set.



Variable	Description
$\bar{n}$	the number of service requests in the training/test data-set
d_i	the collected service requests
c_j	the jth failure mode being monitored
$\bar{f}_i$	manually identified failure mode label for d_i
Φ^{c_j}	classification function for failure mode c_j
$\Phi^{c_j}_{optimal}$	optimal classification function for failure mode c_j
$\hat{f}_i$	predicted label for d_i

Figure 4.5: Machine learning work-flow

Training/Test Data-Set

The training/test data-set consists of $\bar{n}$ service requests randomly sampled from d_i $(= 1, 2, ..., n)$. For each problem description, $\bar{\pi}_i$ $(i = 1, 2, ..., \bar{n})$, in this random sample we associate a failure mode label, $\bar{f}_i$, that best describes the problem being described by the customer. Let $\bar{d}_i$ $(= 1, 2, ..., \bar{n})$ denote the labelled service request:

$$\bar{d}_i = (\bar{\pi}_i, \bar{f}_i). \tag{4.9}$$

The failure mode label $\bar{f}_i$ for each service request $\bar{d}_i$ $(i = 1, 2, ..., \bar{n})$ are manually determined by subject matter experts from the product engineering team. The expert first reads the problem description $\bar{\pi}_i$ and uses the FMEA created in the *External* layer to decide the most appropriate failure mode from the set of failure modes c_j $(j = 1, 2, ..., m)$. Problem description that do not correspond to any failure mode c_j $(j = 1, 2, ..., m)$ are labeled with a "miscellaneous" label. We use c_0 to denote the "miscellaneous" label.

Machine Learning

The labeled service requests $\bar{d}_i$ $(i = 1, 2, ..., \bar{n})$, is the input to machine learning algorithms for the purposes of creating the classification function Φ for mapping customer problem descriptions to failure modes. The first step in applying these algorithms is to transform each text problem description $\bar{\pi}_i$ into a vector of numerical features, $\bar{X}_i$, that can be processed by the machine learning algorithms.

Let V be the set of distinct words used in the set of problem descriptions $\bar{\pi}_i$ $(i = 1, 2, ..., \bar{n})$, v_j be the jth word in V, and $|V|$ be the total number of words in V. We can then represent each service request as a vector of $|V|$ features. The feature vector corresponding to service request $\bar{\pi}_i$ is denoted as $\bar{X}_i$ where the elements $\bar{x}_{ij}$ $(j = 1, 2, ..., |V|)$ are defined as follows:

$$\bar{x}_{ij} \triangleq \begin{cases} 1 & \text{if } v_j \in \bar{\pi}_i \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

The machine learning algorithms uses labeled training instances $(\bar{X}_i, \bar{f}_i)$ to create the classification function Φ. The performance of machine learning algorithms can vary significantly across different data-sets [Caruana and Niculescu-Mizil, 2006]. Therefore, in order to determine the best algorithm for the customer service request data we evaluate three well-known algorithms: naive bayes, decision trees, and support vector machines [Witten and Frank, 2005].

The algorithms are applied using a k-fold cross-validation process [Witten and Frank, 2005] where the training/test set $(\bar{X}_i, \bar{f}_i)$ $(i = 1, 2, ..., \bar{n})$ is partitioned into k equally sized sets of $\frac{\bar{n}}{k}$ (typically $k = 10$). The naive bayes, decision trees, and support vector machines algorithms are then applied as follows:

1. Pick one of these k sets as a test set and use the remaining $(k - 1)$ sets as the training sets.

2. Use the $k - 1$ training sets as input to the machine learning algorithms in order to learn the classification function Φ. Evaluate the resulting classification function Φ on the test set.

3. Repeat steps 1 and 2, k times, each time with a different test set.

The support vector machines algorithm is described below in order to illustrate the application of these methods in the context of the service request data-set. The support vector machines (SVM) algorithm ([Cortes and Vapnik, 1995], [Hastie et al., 2009]) is a statistical learning method that uses a separating hyper-plane as the decision boundary for classifying a data-set into two groups or classes. SVMs have a strong theoretical basis in structural risk minimization [Vapnik and Chervonenkis, 1971] and generally perform well in high-dimensional feature spaces, such as a text classification, where it is necessary to balance model complexity and over fitting.

Let $\Phi^{c_j}(X_i)$ denote the separating hyper-plane for failure mode c_j. The hyper-plane is given by:

$$\Phi^{c_j}(X_i) = w^{c_j} \cdot X_i + b^{c_j}, \tag{4.11}$$

where w^{c_j} is a weight vector and b^{c_j} is a bias factor.

The value of $\Phi^{c_j}(X_i)$ is defined to be positive for all feature vectors X_i with failure mode label c_j and negative for all other failure modes labels. The weight vector w^{c_j} and bias b^{c_j} are learned from the labeled training instances $\bar{d}_i$ $(i = 1, 2, ..., \bar{n})$.

In order to compute w^{c_j} and b^{c_j} we assign $\bar{\mathcal{Y}}_{ij}$ to be a binary label that corresponds to whether the training instance $(\bar{X}_i, \bar{f}_i)$ is labeled with failure mode label c_j.

$$\bar{\mathcal{Y}}_{ij} \triangleq \begin{cases} +1 & \text{if } \bar{f}_i = c_j \\ -1 & \text{otherwise} \end{cases} \tag{4.12}$$

One of the key ideas in SVMs is that only a subset of the training instances, called support vectors, are used to compute the values for w^{c_j} and b^{c_j}. These support vectors are chosen in order to maximize the margin between the two classes ($\bar{\mathcal{Y}} = +1$ and $\bar{\mathcal{Y}}_{ij} = -1$). By introducing a set of Lagrange multipliers $\alpha_i^{c_j}$ $(i = 1, 2, ..., \hat{n})$ the problem of finding these support vectors can be formulated as a constrained quadratic optimization problem:

$$\text{Maximize} \quad W(\alpha^{c_j}) = \sum_{i=1}^{\bar{n}} \alpha_i^{c_j} - \frac{1}{2}\sum_{i=1}^{\bar{n}}\sum_{k=1}^{\bar{n}} \alpha_i^{c_j}\alpha_k^{c_j}\bar{\mathcal{Y}}_{ij}\bar{\mathcal{Y}}_{kj}\bar{X}_i \cdot \bar{X}_k \tag{4.13}$$

$$\text{Subject to:} \tag{4.14}$$

$$\alpha_i^{c_j} \geq 0,\ (i = 1, 2, ..\bar{n}) \tag{4.15}$$

$$\sum_{i=1}^{\bar{n}} \alpha_i^{c_j}\bar{\mathcal{Y}}_{ij}^{c_j} = 0 \tag{4.16}$$

The support vectors are the set of training instances $(\bar{X}_i, \bar{f}_i)$ for which $\alpha_i^{c_j} > 0$. The weight vector is then computed from the support vectors as follows:

$$w^{c_j} = \sum_{i=1}^{\bar{n}} \alpha_i^{c_j}\bar{X}_i\bar{\mathcal{Y}}_i^{c_j}. \tag{4.17}$$

There are a number of different algorithms for solving the quadratic programming problem in Equation 4.13. In this work we use the Sequential Minimal Optimization algorithm in the Weka workbench [Witten and Frank, 2005].

Evaluation

For each failure mode, c_j, the k-fold cross-validation process produces a set of k classifiers for each machine learning algorithm. Each classifier, Φ^{c_j}, determines if a particular service request d_i should be labeled with failure mode c_j. We evaluate the performance of these classifiers using the unseen test set of the labeled service requests to determine the optimal algorithm, i.e. SVM, for the service request data-set.

Performance of the classification function Φ^{c_j} is evaluated using the $F1$ score [Witten and Frank, 2005] which balances precision (correctness) with recall (completeness). For the ith fold ($i = 1, 2, ..., k$) of failure mode c_j: let $TP_i^{c_j}$ be the number of true positives, $FP_i^{c_j}$ be the number of false positives, $TN_i^{c_j}$ be the number of true negatives, and $FN_i^{c_j}$ be the number of false negatives.

The precision $p_i^{c_j}$ and recall $r_i^{c_j}$ are then given by

$$p_i^{c_j} = \frac{TP_i^{c_j}}{TP_i^{c_j} + FP_i^{c_j}}, \tag{4.18}$$

and

$$r_i^{c_j} = \frac{TP_i^{c_j}}{TP_i^{c_j} + FN_i^{c_j}}. \tag{4.19}$$

The $F1$ score for the classification function Φ^{c_j} is the average of the $F1$ scores over the k-folds [Witten and Frank, 2005]:

$$F1^{c_j} = \frac{1}{k}\sum_{i=1}^{k} \frac{2p_i^{c_j} r_i^{c_j}}{p_i^{c_j} + r_i^{c_j}}. \tag{4.20}$$

Classification

Let $\Phi_{optimal}^{c_j}$ be the classification function with the highest $F1^{c_j}$ score for failure mode c_j. The process for using $\Phi_{optimal}^{c_j}$ ($j = 1, 2, ..., m$) to determine the failure mode label for each service request is as follows:

1. Create a feature vector X_i for each service request problem description π_i ($i = 1, 2, ...n$). The value of x_{ij} is given by Equation 4.10.
2. For each service request d_i ($i = 1, 2, ..., n$) compute $\Phi_{optimal}^{c_j}(X_i)$ ($j = 1, 2, ..., m$).
3. $\hat{f}_i$ is the corresponding failure mode label for d_i. For the SVM classifier, the value

of $\hat{f}_i$ is the failure mode c_j such that the sign of $\Phi^{c_j}_{optimal}(X_i)$ is positive. The value of $\Phi^{c_j}_{optimal}(X_i)$ will be negative for all other values of c_j $j = 1, 2, ..., m$.

The data set collected by the product manufacturer consisted of 100,000 customer service requests from a four month period. Each service request in the data set consisted of: (1) a time-stamp indicating when the service request was opened, and (2) a free-form text problem description provided by the customer.

The free-form text problem description varied from one to two paragraph in length. The collected data set contained support cases involving a wide range of ASA 5505 functionality (e.g. authentication, intrusion detection, fail-over). To address this issue we created a two level classifier. The first level filtered out the support cases that were not related to the fail-over function of the ASA 5505. The second level labeled the fail-over related support cases with the corresponding failure mode from the ten failure modes that were selected to monitor. Support cases involving a failure modes not related to the ten failure modes being monitored were labeled as "Misc".

The classifiers were trained using two sets of 1,000 training instances. The first set was randomly sampled from the 100,000 collected support cases and labeled by the product engineering team with binary label indicating if the support cases was related to the fail-over feature or not. The resulting classifier was then applied to obtain the set of fail-over related support cases. The second set of training instances were randomly sampled from the fail-over related support cases and were labeled with a corresponding failure mode by the product engineering team. The FAST and FMEA models for the ASA 5505 were used as a reference when labeling the fail-over related support cases.

The classification function generated using the support vector machines algorithm outperformed the naive bayes and Decision Tree classification functions by a small margin (Table 4.3). All three algorithms had excellent accuracy (98 - 99 %) for the first level classifier, determining if a service request was related to the fail-over feature.

This was largely due to the word “fail-over” being an excellent predictor if a particular service request was related to the fail-over feature. Accuracy dropped significantly (67 - 72 %) at the second level of classification where the service requests were labeled with a corresponding failure mode.

Classification Algorithm	Level 1 (Fail-over)	Level 2 Accuracy (Failure Mode)
Support Vector Machines	99.6 %	72.3 %
Decision Trees	99.2 %	70.6 %
Naive Bayes	98.4 %	67.6 %

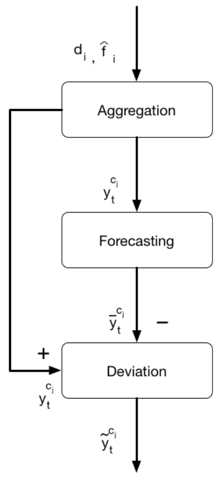
Table 4.3: Evaluation of the support vector machines, naive bayes, and Decision Tree Classifiers

Out of the 100,000 collected support cases, 16,127 were related to the ASA 5505 fail-over feature. The occurrence of the ten failure modes within these 16,127 fail-over related cases was separated into high and low occurrence failure modes. The three most frequent failure modes “configuration sync”, “fail-over link”, and “license”, accounted for over 60% of the total number of fail-over related support cases. In contrast, the less frequent failure modes such as “memory” only accounted for 1% of the total support cases. This large difference between the high-frequency and low-frequency failure modes is partially due to the high-frequency failure modes covering broader problems.

4.5.3 Internal Layer: Time-Series Analysis

The *Internal* layer transforms the labeled service request data into time-series data that can be used for quality monitoring and assessment. For each failure mode we create three different time-series: the actual daily occurrence, the nominal daily occurrence, and the daily deviation between the actual and nominal occurrence. These three time-series are then used at the *Inside-Out* layer to compute the desired quality metrics.

The overall process for creating the actual, nominal, and deviation time-series is shown in the flow chart in Figure 4.6. The aggregation step uses the labels from the *Outside-In* layer to create the time-series of each failure mode‘s daily occurrence. Next, we use time-series analysis in order to forecast the nominal occurrence of each failure mode from the time-series of the failure mode’s actual occurrence. Finally, the deviation for each day is computed by differencing the values for the actual and nominal occurrence.



Variable	Description
c_j	the jth failure mode being monitored
d_i	the ith service request
$\hat{f}_i$	the predicted failure mode label for d_i
$y_t^{c_j}$	occurrence of failure mode c_j on day t
$\bar{y}_t^{c_j}$	nominal occurrence of failure mode c_j on day t
$\tilde{y}_t^{c_j}$	deviation between $y_t^{c_j}$ and $\bar{y}_t^{c_j}$

Figure 4.6: Time-series analysis work-flow

Aggregation

The aggregation process, shown in Figure 4.7, for creating the time-series of each failure mode‘s occurrence consists of two steps: mapping and counting. During the mapping step, each of the n service requests is mapped to a l-by-m matrix corresponding to the l days in the time-horizon of interest and the m failure modes we are monitoring. Let

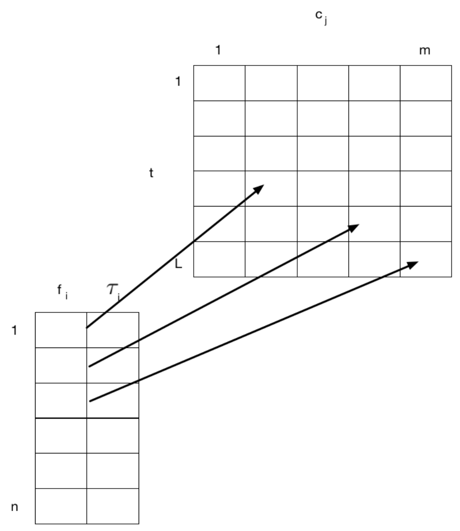


Figure 4.7: Aggregation process for creating the time-series of each failure mode

$D_t^{c_j}$ be the set of service requests for failure mode label c_j and time stamp t. If $\hat{f}_i$ and τ_i are the failure mode label and time-stamp associated with service request d_i, then

$$D_t^{c_j} \triangleq \{d_i : (\hat{f}_i = c_j) \wedge (\tau_i = t)\}. \tag{4.21}$$

We then count the number of service requests associated with failure mode label c_j that were received on day t in order to create the time-series of each failure mode's occurrence. Let $y_t^{c_j}$ be the total number of times failure mode c_j occurred on day t. The value of $y_t^{c_j}$ is given by

$$y_t^{c_j} \triangleq |D_t^{c_j}|, \tag{4.22}$$

where $|D_t^{c_j}|$ is the size of $D_t^{c_j}$.

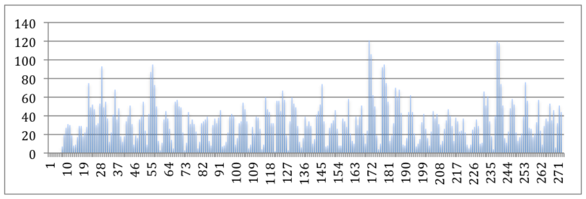


Figure 4.8: Example of the weekly seasonality in the service request time-series

Forecasting

The nominal occurrence of failure mode c_j is the typical number of service requests with label $\hat{f}_i = c_j$ we would expect for any particular day t based on the historical occurrence of that failure mode. Figure 4.8 shows the actual occurrence time-series of a representative failure mode. In this time-series we observe that there is a recurring weekly pattern in the number of service requests for the ASA 5505.

The data has a weekly cyclic pattern of seasonality for the following reason. During the work week (Monday - Friday) customers generally report problems the same day that they occur. However, failures that occur on the weekend (Saturday and Sunday) are not reported until the beginning of the following week (Monday). Consequently, there is a weekly cyclic pattern in the collection of SRs as seen in Figure 4.8.

Let $\bar{y}_t^{c_j}$ denote the forecast for the nominal occurrence of failure mode c_j on day t. Since the failure mode time-series data has seasonality we use a level, trend, and seasonality corrected exponential smoothing model ([Bowerman et al., 2004], [Chopra, 2007]) for creating the forecast $\bar{y}_t^{c_j}$ $(t = 1, 2, ..., l)$. The exponential smoothing method was selected over other well-known forecasting techniques such as ARIMA [Bowerman et al., 2004] because its simplicity makes it relatively straightforward to implement and tune for a data-set. Furthermore, exponential smoothing has been shown to be effective in BioSurveillance time-series forecasting problems [Lotze and Shmueli, 2009], which are similar to our quality monitoring problem.

The exponential smoothing method [Chopra, 2007] consists of three factors: level, trend, and seasonality. The forecast for day t is given by:

$$\bar{y}_t^{c_j} = (L_{t-1}^{c_j} + T_{t-1}^{c_j}) S_{t-1-p}^{c_j}. \tag{4.23}$$

Note that in the reference [Chopra, 2007] the forecast equation is given for the time period $t+1$, however, in this work we have adjusted the forecasting equations to the time period t. This adjustment was made because we are using the exponential smoothing methods to estimate the nominal component for each time period t ($t = 1, 2, ..., l$); we are not forecasting into the future, i.e. $t+1$.

The estimate $L_t^{c_j}$ for the level, the estimate $T_t^{c_j}$ for the trend, and the estimate $S_t^{c_j}$ for the seasonality at time t for failure mode c_j are then updated based on $y_t^{c_j}$ as follows:

$$L_t^{c_j} = \alpha^{c_j} \frac{y_t^{c_j}}{S_{t-p}^{c_j}} + (1-\alpha^{c_j})(L_{t-1}^{c_j} + T_{t-1}^{c_j}) \tag{4.24}$$

$$T_t^{c_j} = \beta^{c_j}(L_t^{c_j} - L_{t-1}^{c_j}) + (1-\beta^{c_j}) T_{t-1}^{c_j} \tag{4.25}$$

$$S_t^{c_j} = \gamma^{c_j} \frac{y_t^{c_j}}{L_t^{c_j}} + (1-\gamma^{c_j}) S_{t-p}^{c_j} \tag{4.26}$$

where p is the periodicity of the seasonality associated with the failure mode time-series and α^{c_j}, β^{c_j}, γ^{c_j} are smoothing constants that are used to minimize the forecast error.

The process for using the exponential smoothing equations to forecast the nominal for a particular failure mode c_j consists of three steps as follows:

1. **Initialization**: set the initial the values for level, trend, and seasonal factors.
2. **Tuning**: determine the optimal values for the smoothing constants in order to minimize the forecast error.
3. **Forecasting**: compute $\bar{y}_t^{c_j}$ over the time-horizon of interest.

Each of the three steps—initialization, tuning, and forecasting—are discussed in detail below.

Initialization

The initial level and trend values are computed directly from the actual time-series for failure mode. Let $L_0^{c_j}$ and $T_0^{c_j}$ denote the initial values for the level and trend of failure mode c_j. These values are set using least squares regression formula [Bowerman et al., 2004] given below:

$$L_0^{c_j} = \frac{\sum_{t=1}^{l} y_t^{c_j} - T_0^{c_j} \sum_{t=1}^{l} t}{l} \tag{4.27}$$

$$T_0^{c_j} = \frac{l \sum_{t=1}^{l} (y_t^{c_j})(t) - (\sum_{t=1}^{l} y_t^{c_j})(\sum_{t=1}^{l} t)}{l(\sum_{t=1}^{l}(t^2) - (\sum_{t=1}^{l} t)^2)} \tag{4.28}$$

The initialization of the seasonal factors requires that we compute the average seasonal factor for each of the p days in the recurring weekly cyclic pattern. These initial seasonal factors are denoted as $S_t^{c_j}$ $(t = 1-p, 2-p, ..., 0)$. When computing these average seasonal factors we use the simplifying assumption that all of the failure modes have the same reporting delay and, therefore, have the same seasonal factors. This simplification enables us to compute one set of these average seasonal factors from the aggregated occurrence of all failure modes and then use them for each failure mode c_j $(j = 1, 2, ..., m)$.

Let y_t be the aggregated occurrence of all m failure modes at time t:

$$y_t \triangleq \sum_{j=1}^{m} y_t^{c_j}. \tag{4.29}$$

We then deseasonalize y_t using an averaging process [Chopra, 2007] that removes the recurring cyclic fluctuations. Let y'_t denote the deseasonalized value of y_t. When p is odd (in our case $p = 7$ corresponding to weekly seasonality), y'_t is given by Equation 4.30 below:

$$y'_t = \sum_{i=[t-\lfloor p/2 \rfloor]}^{[t+\lfloor p/2 \rfloor]} \frac{y_i}{p}. \tag{4.30}$$

Since this averaging process requires $\frac{p}{2}$ days of historical data in order to calculate the deasonalized value of any particular day, we compute y'_t starting after the first complete seasonal cycle ($t = p + 1, p + 2, ..., l$).

We then regress y'_t ($t = p + 1, p + 2, ..., l$) [Bowerman et al., 2004] to obtain the estimates for the level and trend of the deseasonalized time-series:

$$L' = \frac{\sum_{t=p+1}^{l} y'_t - T' \sum_{t=p+1}^{l} t}{l - p} \tag{4.31}$$

$$T' = \frac{(l-p) \sum_{t=p+1}^{l} (y'_t)(t) - (\sum_{t=p+1}^{l} y'_t)(\sum_{t=p+1}^{l} t)}{(l-p)(\sum_{t=p+1}^{l} (t^2) - (\sum_{t=p+1}^{l} t)^2)} \tag{4.32}$$

The estimate for the deseasonalized occurrence, denoted as $\bar{y}'_t$, is then obtained from the line of regression given below:

$$\bar{y}'_t \underset{(4.32,4.31)}{=} L' + T't. \tag{4.33}$$

The seasonal factor at time t, denoted as $\bar{S}_t$, is the ratio of the actual occurrence y_t

to regressed deseasonalized occurrence $\bar{y}'_t$:

$$\bar{S}_t \underset{4.33}{\triangleq} \frac{y_t}{\bar{y}'_t}. \tag{4.34}$$

The average seasonal factors, S_t $(t = 1-p, 2-p, ..., 0)$, are then computed as follows:

$$S_t \underset{4.34}{=} \frac{1}{\lfloor l/p \rfloor} \sum_{i=1}^{\lfloor l/p \rfloor} \bar{S}_{(t+ip)}. \tag{4.35}$$

Finally, because we are using the same set of seasonal factors for all failure modes c_j,

$$S_t^{c_j} = S_t \ (t = 1-p, 2-p, ..., 0)(j = 1, 2, ..., m). \tag{4.36}$$

Tuning

The smoothing constants α^{c_j}, β^{c_j}, γ^{c_j} are used to tune the exponential smoothing method for a particular data-set. These constants are set separately for each failure mode c_j $(j = 1, 2, ..., m)$ and are optimized in order to minimize the forecast error.

The smoothing constants are optimized in three steps. First, we set each smoothing constant to a seed value between (0,1), e.g. 0.05, and compute the forecasts for $t = 1, 2, ..., l$. We then compute the forecast error, ϵ^{c_j}, for $t = 1, 2, ..., l$:

$$\epsilon^{c_j} = \sum_{t=1}^{l} (\bar{y}_t^{c_j} - y_t^{c_j})^2. \tag{4.37}$$

The values of α^{c_j}, β^{c_j}, γ^{c_j} are then adjusted to minimize ϵ^{c_j}. This problem can be formulated as a non-linear programming problem and solved using Microsoft Excel, R

[R Development Core Team, 2008], or a number of other software tools.

Forecasting

It is important to note that we are using forecasting to create an estimate of $y_t^{c_j}$ $(t = 1, 2..., l)$, we are not forecasting future values past the time-horizon of interest $(t = 1, 2..., l)$. The process for creating these estimates is as follow:

1. Compute $\bar{y}_t^{c_j}$ for $t = 1$ using Equation 4.23 with initial values for $L_0^{c_j}$, $T_0^{c_j}$, and $S_{1-p}^{c_j}$.
2. Update the values for $L_t^{c_j}$, $T_t^{c_j}$, and $S_t^{c_j}$ for $t = 1$ using Equations 4.24, 4.25, and 4.26 with the optimized values for smoothing parameters and the actual occurrence $y_t^{c_j}$.
3. Repeat Steps 2 and 3 for $t = 2, 3, ..., l$.

Deviation

$\tilde{y}_t^{c_j}$ is the deviation between the actual and the nominal (expected) values of failure mode c_j on day t:

$$\tilde{y}_t^{c_j} \underset{(4.22,4.23)}{\triangleq} y_t^{c_j} - \bar{y}_t^{c_j}. \tag{4.38}$$

When the value of the deviation is positive it means that there were more service requests for that particular day then would be expected. Likewise, when the deviation is negative, it means that there were less service requests then expected.

4.5.4 Inside-Out layer: Quality Metrics

The *Internal* layer provides the failure-mode time-series for quality monitoring. The problem addressed at the *Inside-Out* layer is computing the quality monitoring metrics from these failure-mode time-series. In this work we compute three sets of metrics: static, nominal, and deviation. These metrics are then used at the *Outer* layer to assess product quality.

Figure 4.9 shows the process for computing the static, nominal, and deviation metrics. The static and nominal metrics for a particular failure mode c_j $(j = 1, 2, .., m)$ are computed directly from their respective time-series. The deviation metrics for a particular failure mode c_j $(j = 1, 2, .., m)$ are computed using the total number of days where there is a significant large value in the failure mode‘s deviation time-series. In order to determine if any particular value in the deviation time-series is significantly large, we use a Shewart control chart applied to the time-series.

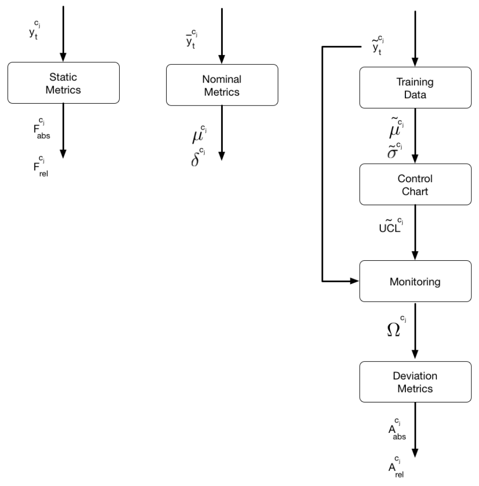


Figure 4.9: Inside-Out layer: computation of the quality metrics

Variable	Description
c_j	the jth failure mode being monitored
$y_t^{c_j}$	occurrence of failure mode c_j on day t
$\bar{y}_t^{c_j}$	nominal (forecasted) occurrence of failure mode c_j on day t
$\tilde{y}_t^{c_j}$	deviation between $y_t^{c_j}$ and $\bar{y}_t^{c_j}$
$\tilde{UCL}^{c_j}$	upper control limit for deviation of c_j
Ω^{c_j}	set of days with abnormal deviation for c_j
$F_{abs}^{c_j}$	absolute occurrence of c_j
$F_{rel}^{c_j}$	relative occurrence of c_j
μ^{c_j}	mean occurrence of c_j
δ^{c_j}	growth in occurrence of c_j
$A_{abs}^{c_j}$	absolute number of days where $\tilde{y}_t^{c_j}$ was abnormal
$A_{rel}^{c_j}$	relative number of days where $\tilde{y}_t^{c_j}$ was abnormal

Static Metrics

We use two static metrics to characterize the total occurrence of a failure mode c_j $(j = 1, 2, 3, ..., m)$ over the time horizon of l days: absolute occurrence and relative occurrence.

We compute the absolute occurrence, $F_{abs}^{c_j}$, as the cumulative or total occurrence of failure mode c_j over the time horizon of l days:

$$F_{abs}^{c_j} \triangleq \sum_{t=1}^{l} y_t^{c_j}. \tag{4.39}$$

In order to compare the absolute occurrence of different failure modes we normalize $F_{abs}^{c_j}$. Let F_{abs} be the cumulative or total number of occurrence of all m failure modes over the time horizon of l days:

$$F_{abs} \triangleq \sum_{j=1}^{m} \sum_{t=1}^{l} y_t^{c_j}, \tag{4.40}$$

where $y_t^{c_j}$ is given by Equation 4.22.

$F_{rel}^{c_j}$, the normalized occurrence of failure mode c_j relative to the total occurrence of all other failure modes over the time horizon, is given by:

$$F_{rel}^{c_j} \triangleq \frac{F_{abs}^{c_j}}{F_{abs}} \underset{(1,2)}{=} \frac{\sum_{t=1}^{l} y_t^{c_j}}{\sum_{j=1}^{m} \sum_{t=1}^{l} y_t^{c_j}}. \tag{4.41}$$

Nominal Metrics

The nominal metrics characterize the typical daily occurrence of each failure. For each failure mode c_j, we compute the mean daily occurrence and the average growth (trend) of the daily occurrence from the nominal time-series $\bar{y}_t^{c_j}$ $(t = 1, 2, 3, ..., l)$.

Let μ^{c_j} denote the mean or average of the nominal daily occurrence of failure mode c_j over the time-horizon of l days:

$$\mu^{c_j} \triangleq \frac{\sum_{t=1}^{l} \bar{y}_t^{c_j}}{l}, \tag{4.42}$$

where $\bar{y}_t^{c_j}$ is given by Equation 4.23.

Let δ^{c_j} denote the rate of change (growth) of the nominal daily occurrence in failure mode c_j over the time horizon of l days. We compute δ^{c_j} as the slope obtained by regressing $\bar{y}_t^{c_j}$ $(t = 1, 2, ..., l)$ as follows:

$$\delta^{c_j} \triangleq \frac{l \sum_{t=1}^{l} (\bar{y}_t^{c_j})(t) - (\sum_{t=1}^{l} \bar{y}_t^{c_j})(\sum_{t=1}^{l} t)}{l(\sum_{t=1}^{l} (t^2) - (\sum_{t=1}^{l} t)^2)}. \tag{4.43}$$

Deviation Metrics

The deviation metrics characterize how the actual occurrence of each failure mode deviates from the nominal. If $\tilde{y}_t^{c_j}$ $(t = 1, 2, 3, ..., l)$ is the time-series of the deviation between the actual occurrence and nominal (expected) occurrence of failure mode c_j, then we first determine the set of days for which the value of $\tilde{y}_t^{c_j}$ is abnormally large as determined by the control chart described below.

We use a **Shewhart Control Chart** [Lawson and Erjavec, 2001] to set the upper threshold for distinguishing between normal and abnormal values in the deviation time-series $\tilde{y}_t^{c_j}$ $(t = 1, 2, ..., l)$. Typically a Shewart Control Chart consists of a center line that represents the mean value of the time-series and two other horizontal lines, called the upper control limit (UCL) and the lower control limit (LCL) which define the upper and lower threshold for normal values. For the purposes of quality monitoring where we are interested in detecting increases in the deviation, i.e. more failures then the nominal, we only compute the UCL.

The value for the UCL threshold is set based on the mean and standard deviation of the time-series during a training period where there are no days where the deviation time-series is abnormally large. This training period is manually identified by subject matter experts who can assess if a particular deviation value, $\tilde{y}_t^{c_j}$, is abnormal.

Let $\tilde{\mu}^{c_j}$ and $\tilde{\sigma}^{c_j}$ denote the mean and standard deviation, respectively, of the deviation time series $\tilde{y}_t^{c_j}$ during the training period. Then, $U\tilde{C}L^{c_j}$, the upper control limit for the deviation time-series $\tilde{y}^{c_j}$ is given by

$$U\tilde{C}L^{c_j} \triangleq \tilde{\mu}^{c_j} + k\tilde{\sigma}^{c_j}. \tag{4.44}$$

The value of k is set so that the probability that an observed value would fall outside the control limits is very small. In this work we use a three standard deviation limit $(k = 3)$ for the upper control limit which provides a 99.865 % probability that a value larger than $U\tilde{C}L^{c_j}$ is abnormal [Lotze and Shmueli, 2009].

Ω^{c_j} is the set of days where deviation of c_j is larger than the corresponding upper control limit $U\tilde{C}L^{c_j}$:

$$\Omega^{c_j} \triangleq \{t | \tilde{y}_t^{c_j} > U\tilde{C}L^{c_j}\}. \tag{4.45}$$

The absolute deviation, $A_{abs}^{c_j}$, is the total number of days in the set Ω^{c_j}:

$$A_{abs}^{c_j} \triangleq |\Omega^{c_j}|. \tag{4.46}$$

$A_{rel}^{c_j}$ is the number of days where failure mode c_j has a significant deviation normalized over the time horizon l:

$$A_{rel}^{c_j} \triangleq \frac{A_{abs}^{c_j}}{l}. \tag{4.47}$$

Table 4.4 shows the complete set of static, nominal, and dynamic quality metrics for the ten critical ASA 5505 fail-over related failure modes. The use of these metrics to monitor and assess the quality of the ASA 5505 fail-over feature is discussed in Section 4.6. In order to illustrate the metrics for the ASA 5505, we will discuss a subset three high-frequency failure modes ("license", "VPN", "configuration sync") and one low frequency failure mode ("software upgrade").

The mean occurrence varied significantly between the ten fail-over failure modes being monitored. The four high-frequency failure modes had a mean occurrence between seven and ten failures per a day. The other six, low-frequency, failure modes occurred approximately one time per a day. The growth for all ten failure modes was very small during the monitoring period (between 0.0103 and -0.003 failures per day). Figure 4.10 illustrates the mean occurrence and growth metrics for three of the high-frequency failure modes ("license", "VPN", "configuration sync") and one low-frequency failure mode ("software upgrade"). The small value for the growth metric indicates that the quality of the ASA 5505 fail-over feature is not changing over time.

Figure 4.10 illustrates the deviation time-series and corresponding control chart for four ASA 5505 failure modes ("license", "VPN", "configuration sync", and "software

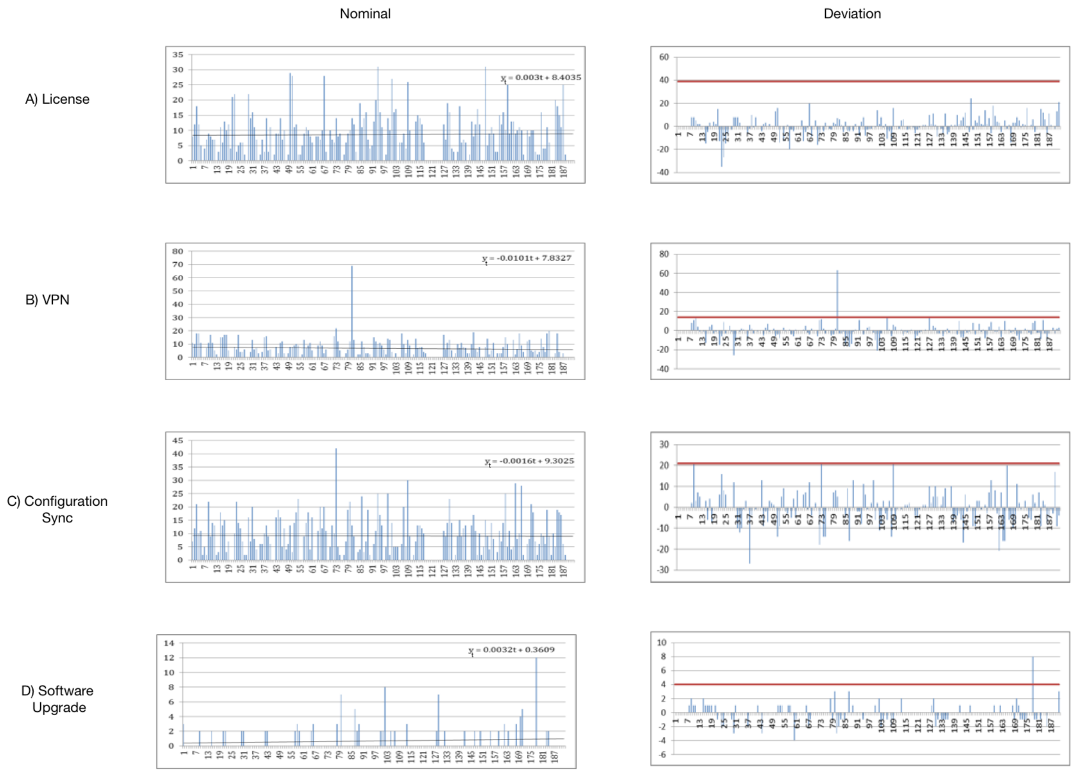


Figure 4.10: Nominal and Deviation metrics for the (A) ASA 5505 license, (B) VPN, (C) configuration sync, and (D) software upgrade failure modes

upgrade"). The majority of the deviations in product quality (18 out of the 22 total occurrence of deviation across all failure modes) were for failure modes with a mean occurrence of approximately one failure mode per a day such as "fail-over communications" and "both active". For these failure modes, it is likely that the sparseness of the historical data resulted in an artificially low control chart threshold. None of the failure modes being monitored had a relative deviation value larger than 4% of the total number of days monitored.

4.5.5 Outer Layer: Quality Assessment

The *Outer* layer is where product quality is assessed based on the metrics from the *Inside-Out* layer. This assessment consists of two parts. First, we develop a set of guidelines for using the metrics to assess product quality. For each set of quality metrics, we then apply the guidelines to the monitoring results from *Inside-Out* layer and determine the necessary product design, product development, and product delivery actions, if any, that need to be taken for each sub-system in order to improve product quality.

We now illustrate the implementation of the *Outer* layer for the problem of monitoring and assessing product quality for the ASA 5505 network security device. The static quality metrics enable the assessment of the overall quality of the different product sub-systems/components. In product design, the assessment of the static metrics enables the identification of low-quality product sub-systems/components that need to be potentially re-designed in future products. During product development, the assessment of the static metrics can help focus product testing and quality assurance processes on the most frequently occurring failure modes and/or product sub-systems.

The assessment process for the static quality metrics is as follows:

1. Rank the failure modes in descending order of relative occurrence.
2. Use the ranking to organize the failure modes into high-occurrence and low-occurrence groups.
3. Determine an appropriate threshold for the RPN values in the high-occurrence group. This threshold should be set based on historical failure mode data from similar products.
4. Apply the threshold to identify the subset of critical failure modes in high-occurrence group.

The assessment of the nominal metrics has two primary applications related to product delivery. First, we want to identify failure modes where the nominal occurrence is increasing over time. This would indicate a systematic problem that needs to be resolved. Second, we want to use the nominal metrics in order to forecast the daily frequency of occurrence for each failure mode. These forecasts would enable better allocation of product delivery resources such as technical support teams.

The assessment process for the nominal metrics is as follows:

1. Rank the failure modes by mean occurrence.
2. Use the ranking to organize the failure modes into high-occurrence and low-occurrence groups. (Note: these groups should be consistent with the groups created during the assessment of the static metrics.)
3. Determine an appropriate threshold for the RPN values in the high-occurrence group. This threshold should be set based on historical failure mode data from similar products.
4. Failure modes that have a large negative trend (growth rate) can safely be ignored regardless of the RPN value. Failure modes with a small negative or neutral trend and a RPN above the threshold be investigated by the product delivery teams. Failure modes with large positive trend should be investigated by the product delivery teams regardless of the RPN value.

The assessment of the deviation metrics enables the detection of abnormal changes in the frequency of occurrence of product failure modes. These abnormalities are often early indications of emerging product quality problems such as software defects. Detecting these emerging issues before they impact a large number of customers can be used to improve the effectiveness and efficiency of product delivery processes.

The assessment process for the deviation metrics is as follows:

1. Rank the failure modes by absolute deviation.

2. Use the ranking to organize the failure modes into high-occurrence and low-occurrence groups.

3. Determine an appropriate threshold for the RPN values in the high-occurrence group. This threshold should be set based on historical failure mode data from similar products.

4. Apply the threshold to identify the subset of critical failure modes in high-occurrence group.

The results of applying these guidelines to assess the quality metrics from Table 4.4 is discussed in Section 4.6.

4.6 Results: Quality Monitoring and Assessment for the ASA 5505 Network Security Product

In this section we describe the results of quality monitoring and assessment for the ASA 5505 network security product. The results are organized into two parts. First, we summarize the absolute, nominal, and deviation quality metrics for the ASA 5505. We then assess the quality of the ASA 5505, using the computed quality metrics, and discuss the potential application of the assessment to improve product design, development, and delivery. The assessment results directly follow from the implementation of the assessment guidelines developed in the *Outer* layer of the Product Quality Monitoring Process Methodology (Section 4.5.5).

4.6.1 Monitoring ASA 5505 Quality

Product quality is monitored using a set of metrics computed from the 100,000 collected customer service requests. Table 4.4 shows the results of computing the quality metrics for the ten selected failure modes related to the ASA 5505 fail-over feature. For a particular failure mode c_j, the absolute occurrence metric $F_{abs}^{c_j}$ measures the total number of times c_j has occurred (Equation 4.39) in the collected data set D. The relative occurrence metric $F_{rel}^{c_j}$ measures the occurrence of c_j relative to the occurrence of all failure modes (Equation 4.41).

For the ASA 5505 the absolute and relative occurrence metrics show that the occurrence of the ten monitored failure modes was unbalanced and not uniformly distributed. The five high-frequency failure modes ("configuration sync", "no switch-over", "fail-over link", "license", and "VPN") each occurred more than 1,000 times and accounted for the majority of the ASA 5505 fail-over related failures. The remaining five low-frequency failure modes had a low absolute occurrence (between 100 and 250 times during the four month in which the failure modes were monitored). We also found that a significant number (56 %) of the fail-over related service requests, labeled as "miscellaneous" in Table 4.4, that did not correlate to one of the ten critical failure modes we were monitoring.

The mean occurrence metric μ^{c_j} measures the mean daily occurrence of failure mode c_j (Equation 4.42). The growth metric, δ^{c_j}, measures the mean rate of change of the daily occurrence of c_j (Equation 4.43). For the ASA 5505, the mean occurrence varied significantly between the ten critical fail-over failure modes. The five high-frequency failure modes had a μ^{c_j} value between seven and ten failures per a day. The other five, low-frequency, failure modes occurred approximately once per day. The rate of change for all ten failure modes was constant over time and exhibited little change (δ^{c_j} was between 0.0103 -0.003 failures per day). The small value for δ^{c_j} suggests that the quality of the ASA 5505 fail-over sub-system is not changing over time.

For a particular failure mode c_j, the absolute deviation metric, $A_{abs}^{c_j}$, measures the total number of days where the deviation component of c_j is larger than the threshold defined by the upper control limit, UCL^{c_j} (Equation 4.44). The relative deviation metric, $A_{rel}^{c_j}$, measures the percentage of the time that the deviation component of c_j is

Failure Mode	Absolute Occurrence ($F_{abs}^{c_j}$)	Relative Occurrence ($F_{rel}^{c_j}$)	Mean Occurrence (μ^{c_j})	Growth (δ^{c_j})	Absolute Deviation ($A_{abs}^{c_j}$)	Relative Deviation ($A_{rel}^{c_j}$)
Both Active	258	1.58 %	0.78	-0.0002	8	4.1 %
Configuration Sync	1721	10.67 %	9.30	-0.0016	0	0 %
No Switch-over	1498	9.28 %	8.25	-0.0035	3	1.5 %
Fail-over Communication	206	1.26 %	1.1044	-9E-05	6	3.1 %
Fail-over Link	1731	10.73 %	8.24	0.0103	4	2 %
Both Active	143	0.88 %	0.78	-0.0002	0	0 %
Software Upgrade	158	0.96 %	0.361	0.0032	2	1 %
License	1690	10.47 %	8.40	0.003	0	0 %
Memory	92	0.6 %	0.8048	-0.0033	0	0 %
VPN	1333	8.26 %	7.8327	-0.0101	2	1 %
Miscellaneous	9181	56.9 %	51.73	-0.0031	0	0 %
Aggregate	16127	100 %	88.56	-0.029	0	0 %

Table 4.4: Quality metrics for the ASA 5505 fail-over feature

larger than this threshold (Equation 4.47). During the four month monitoring period the values for absolute occurrence metric for the ten ASA 5505 failure modes ranged from zero to eight days where the deviation was significantly large. The majority of the deviations in product quality (18 out of the 22 total counts of deviation across all failure modes) were for failure modes, such as "fail-over communications" and "both active", with a mean occurrence of approximately one failure mode per a day. For these failure modes, it is likely that the sparseness of the historical occurrence data resulted in an artificially low UCL^{c_j} threshold. None of the failure modes being monitored had a relative deviation larger than 4% of the total number of days monitored.

4.6.2 Assessment of ASA 5505 Quality

The assessment component of the Product Quality Monitoring and Assessment Process Methodology (PQMAPM) is based on applying the guidelines to quality metrics. The first set of guidelines, related to the static quality metrics, assess the overall quality of the different product sub-systems. We start by identifying the set of failure modes with a high absolute occurrence. The five high-occurrence failure modes are as follows: "configuration sync", "no switch-over", "fail-over link", "license", and "VPN". We then set a threshold for the RPN value from the product's Failure Modes and Effects Analysis (FMEA) and absolute occurrence and identify the subset of critical failure modes. This threshold is typically determined using historical data for absolute and relative occurrence of failure modes for similar products. For this particular problem we did not have this data available and the product experts estimated a threshold of 100 for the RPN value and a threshold value of 1,500 for the absolute occurrence. The application of this threshold to the high-occurrence failure modes identified two critical failure modes: "configuration sync" and "fail-over link".

A second important application of the absolute and relative metrics is assessing the completeness of the critical failure modes selected in the *External* layer of the

PQMAPM. When the relative occurrence of the“miscellaneous” failure mode is large, it is useful to work with the product subject matter experts to refine the FMEA and identify new failure modes that need to be monitored. The assessment of ASA 5505 determined that the “miscellaneous” failure mode had an larger than expected absolute and relative occurrence. This suggests that it would be useful to re-evaluate the product’s FMEA and expand the number of critical failure modes until the relative occurrence was a smaller proportion of the total number of failure modes.

The second set of guidelines, for assessing the nominal metrics, characterize the occurrence of different product failure modes over time and can be used to improve product delivery. In the ideal scenario the values for the mean occurrence and growth metrics for each failure mode would be benchmarked against historical data for similar products in order to identify potential issues. Since historical data was not available for this particular problem we used the product experts to manually review the nominal quality metrics and determine the set of critical failure modes based on the mean occurrence and growth metric values. The product experts’ assessment was that there was no critical failure modes with respect to the nominal metrics because the failure modes with a large mean occurrence (“configuration sync”, “no switch-over”, “fail-over link”, “license”, and “VPN”) had a growth metric value of close to zero. The mean occurrence and growth metrics can also be used in the Exponential Smoothing model (Equation 4.23) to forecast the daily service request volume for each failure mode. These forecasts could be used to improve resource allocation by ensuring that the necessary number of support engineers were available for the anticipated service request volume.

The third set of guidelines address the deviation metrics and enable the detection of abnormal product behavior that indicate possible emerging quality issues, e.g. a serious defect in a new software release. From the results of Table 4.4, there are three failure modes with a large value for the absolute deviation metric: “both active”, “fail-over communications”, and “fail-over link”. Similar to the assessment of the static metrics, we then apply a threshold based the RPN value and absolute occurrence. Product experts estimated a threshold of 100 for the RPN value and a threshold value of 5 for the absolute deviation. Using the absolute deviation threshold, “fail-over communications” is a critical failure mode. Further investigation by product experts determined that there were no specific quality issues related to the “fail-over communication” failure mode.

Using the FAST diagram for the ASA 5505 we then mapped the critical failure modes to the corresponding product sub-systems. The “configuration sync’ and “fail-over communication” failure modes both mapped to the stateless fail-over sub-system; while the “fail-over link” failure mode mapped to the fail-over link sub-system. The presence of two critical failure modes in the “stateless fail-over” sub-system suggests that it would be useful for the product design team to potentially redesign this sub-system in future network security products. Since the assessment only identified a single failure mode for the fail-over link sub-system, it is may not be necessary to redesign this sub-system.

For this problem, the quality monitoring and assessment results show that there are no serious quality issues with the ASA 5505 fail-over feature and, therefore, no further actions need to be taken. In general product quality information can be used by the product manufacturer to improve product design, development, and delivery. In product design, the assessment of the static metrics enabled the identification of low-quality product sub-systems/components that need to be potentially re-designed in future products. During product development, the assessment of the static metrics can help focus testing processes on the most frequently occurring failure modes and/or product sub-systems. During product delivery, the assessment enables the identification of dominant failure modes that are growing in frequency as well as the detection of any emerging quality issues.

4.7 Conclusions

For the product quality monitoring and assessment problem, the five representational layers of the Integrated Meta-Representational Model (IMRM) enabled the clear separation of the following issues: domain knowledge representation (*External* layer), integration of the domain knowledge with data mining (*Outside-In* layer) and time-series analysis (*Internal* layer), and the KE software environment. One of the key results of explicitly separating these issues was the identification of Engineering Design methods, a domain not typically considered in knowledge engineering problems, in order to formally model the product's failure modes. Once the data was structured by failure mode, it was relatively straight-forward to calculate quality metrics that were immediately useful to the product engineering teams.

Traditionally data mining is a data-driven process where algorithms are applied to data and the resulting patterns are interpreted for novelty and usefulness. One of the frequently cited limitations of this approach is that the extracted patterns are generally based on correlation between variables and do not incorporate causal relationships Glymour et al. [1997]. This limitation is particularly an issue in engineering applications where cause and effect, e.g. understanding the factors that caused a particular failure mode to occur, plays an important role in making knowledge actionable.

The incorporation of domain knowledge into the data mining process is one potential way to address the issue of causality and ensure that knowledge engineering produces results that are meaningful to the problem under consideration. One of the conclusions from our work is that when attempting to model domain knowledge in technical domains, e.g. computer networking, it is beneficial to first model the engineering problem under consideration. Most engineering systems have explicit, or derivable, domain models which have causality built into their structure. Therefore, the use of these domain models to drive the knowledge engineering process enables causality to be naturally incorporated. This use of causality is in contrast to statistical based approaches Silverstein et al. [2000] which are computationally expensive and limited in the causal models they are capable of discovering.

5 A Statistical Design of Experiments Approach to Machine Learning Model Selection: Theory and Application to Predicting Customer Service Escalation

In this chapter we address the application of the representation-based framework, developed in Chapter 3, to the knowledge engineering problem of predicting which customer service requests will need to be escalated in priority. This problem involves of the more general knowledge engineering research issue of selecting of the best machine learning model for the application. Generally model selection problem has been addressed by optimizing with respect to the internal aspects of the model, namely the learning algorithm and associated hyper-parameters. However, in real-world engineering applications, there are typically a number of other factors that are external to the learning algorithm which have a significant impact of the performance of the model. For example, real-world machine learning problems typically begins with a lengthy pre-processing step in which we select the relevant features, normalize, and sub-sample the data before the learning algorithm is applied.

In this chapter we develop a simple and effective approach based on statistical Design of Experiments (DOE) for optimizing these external parameters in combination with the learning algorithm. In the DOE approach we treat each external model parameter, including the learning algorithm, as a experimental factor. We then design a set of experiments using orthogonal arrays to efficiently explore the experimental space and determine the settings that produce the optimal model with respect to a desired performance metric. We demonstrate the utility of the DOE approach using the problem of predicting if a particular customer support case will need to be escalated in priority in order to be resolved in a timely manner. Experimental results show improvement in the optimized classification model‘s performance with respect to the objective of minimizing total cost as well as standard metrics including accuracy, f-measure, and g-mean.

The chapter is organized as follows. Section 5.2 formalizes the problem of model selection and illustrates some of the key issues using an example problem from the computer networking services domain. Section 5.3 discusses existing work related to model selection and motivates the need for a systematic approach to building and testing different machine learning models. Section 5.4 describes our approach to model selection based on using planned experiments. Section 5.5 demonstrates the application of the statistical Design of Experiments (DOE) method to a machine learning problem in the computer networking domain. Section 5.6 presents the results from the knowledge engineering problem application in the computer networking domain. Lastly, in Section 5.7, we discuss the benefits of the using planned experiments, and how the representation-based model enables this approach to be applied to larger set of model selection problems.

5.1 Introduction

The purpose of this section is to describe the machine learning model selection problem, outline some of the research issues involved, and describe our key contributions to these research issues.

5.1.1 Background

An important problem in machine learning is the selection of the settings that produce the best predictive model for a particular application, e.g. predicting customer service request escalation. A key parameter when creating predictive models is the learning algorithm used, e.g. support vector machines, to create the model from the training/test data-set. Furthermore, each learning algorithm has a number of so-called hyper-parameters associated with it that control how the algorithm learns, e.g. the kernel function used in the support vector machines. Existing approaches to model selection have primarily focused on either the selection of the learning algorithm ([Michie et al., 1994], [Brazdil et al., 2003]), the optimization of the hyper-parameters associated with the learning algorithm ([Strijov and Weber, 2010], [Bergstra et al., 2011], [Snoek et al., 2012]), or the combined algorithm selection and hyper-parameter optimization problem [Thornton et al., 2013].

The application of machine learning techniques to real-world engineering problems involves other additional parameters, beyond the learning algorithm and hyper-parameters, which are in general application dependent, and critical to the performance of the model. For example, one issue that frequently comes up in real world problems is an imbalance in the class distribution of the training/test data-set which results in a severe bias in the resulting machine learning model. We can consider the various approaches to addressing this issue—resampling, cost sensitive learning, etc.—as an additional parameter in the machine learning process.

5.1.2 Research Issues

This work addresses the problem of optimizing the combination of the machine learning algorithm and the external parameters for the machine learning process within the context of creating predictive models for engineering applications. This task involves the following research issues:

1. The creation of a rational framework organizing the variety of decisions involved in machine learning model selection in engineering applications.
2. The development of a process methodology for fast and efficient machine learning model selection in engineering applications.
3. The application and evaluation of the process methodology for machine learning model selection using a real-world engineering application.

5.1.3 Contributions

Our contributions to the research issues, described in the previous section, are as follows:

1. We have identified and organized the different types of parameters (decisions) involved in model selection. In the process we have formalized the notion of a set of external parameters for the machine learning model and created a framework that can be used to explicitly structure the engineering application-specific decisions involved in the machine learning process. (See Section 5.2)

2. We have adapted the Taguchi method [Taguchi and Konishi, 1987], from the domain of Statistical Design of Experiments, to create a process methodology for optimizing the external parameters for a particular machine learning model. They key idea in the Taguchi method is the design of an orthogonal array of experiments that spans, in general, a large experimental space of factors and factor levels with a minimal number of experiments. The method consists of three stages. The first stage defines the minimal number of experiments (orthogonal array) to be performed in the experimental space of the factors and the levels associated with each experimental factor. The second stage poses an optimization problem to maximize the s/n ratio based on a user-defined performance metric. The third stage uses Statistical Analysis of the Means (ANOM) to determine the optimal settings for each experimental factor level. The optimal settings are then used to build the machine learning model for the problem of interest. (See Section 5.4)

3. We have demonstrated the utility of the developed process methodology, described above, using an important problem in the computer network domain: predict whether or not a particular customer support case needs to be escalated in priority in order to be resolved in a timely manner. The experimental design for this problem consisted of nine experiments that tested four different machine learning process parameters—feature selection strategy, sampling strategy, learning algo-

rithm, and classifier structure—each with three different levels or settings for the parameters. The Taguchi method resulted in a significant improvement of the model performance with respect to the objective function of minimizing total cost as well as standard machine learning metrics including accuracy, f-measure, and g-mean. (See Section 5.5, 5.6)

5.2 Problem Formulation

All machine learning algorithms have parameters known as hyper-parameters associated with them, for example, the kernel function in the support vector machine algorithm. These parameters can be thought of as "internal" to the model. When creating a machine learning model for engineering problems in complex technical domains there are a number of other additional considerations, "external", to the learning algorithm and associated hyper-parameters that have a significant effect on model performance. For example, real-world data-sets typically require pre-processing (feature selection, normalization, sub-sampling, etc.) before the learning algorithm can be applied. The purpose of this section is to formalize the model selection problem within the context of these external model parameters. To this end, this section is organized into two parts. First, we use the important real-world knowledge engineering problem of predicting service level escalation to develop and illustrate issues that lead to the external model parameters. We then formalize the notion of a machine learning model with external parameters and pose the model selection problem in this context.

Consider a customer support center that provides technical support for computer networking products. Once a customer case is received by the support center, it is typically worked on by the same team of engineers until the problem is resolved. However, for a small percent of cases it is necessary to escalate the service request in priority and, then re-route it to a new support team in order to resolve the customer‘s problem in a timely manner. The process of escalating a case after it has already been routed to a support team is typically very expensive and resource intensive. Proactively predicting which service requests are most likely to be escalated, and, then escalating immediately would significantly reduce the cost associated with the escalation process.

For this problem, the desired machine learning model, denoted Φ, takes a customer service request as input and outputs a binary decision for whether the service request should be escalated in priority. Let X denote the particular customer service request of interest and $y \in \{-1, +1\}$ denote a binary prediction, indicating whether the service request should be escalated ($y = +1$), or not ($y = -1$). The machine learning model, Φ, is defined as the mapping function shown in Equation 5.1.

$$\Phi : X \rightarrow y \tag{5.1}$$

This mapping function, Φ, is generated by applying a machine learning algorithm, e.g. support vector machines, to a data-set of historical service requests where we know the corresponding escalation label for each service request. The process of applying the machine learning algorithm, which we will call the machine learning process, involves a large number of problem-specific decisions that need to be made by the knowledge engineer. For example, some of the decisions involved in the service request escalation problem are as follows:

1. **Feature Selection**: Each service request contains over a hundred different features or attributes; however, many of these features, e.g. customer address, are not relevant for predicting whether a service request needs to be escalated. We therefore need to determine the best method for selecting the subset of features to use when constructing the model.

2. **Asymmetrical Misclassification Cost**: The misclassification costs associated with each service request are not symmetrical. A service request that is not

escalated when it should have been escalated will have a significantly higher cost to the organization compared to a service request that is incorrectly escalated when it should have not been. We therefore need to select the method that optimizes the objective of minimizing the total escalation cost.

3. **Class Imbalance in the Data-set**: The historical data-set of service requests is unbalanced, i.e. the majority of the historical service requests did not require escalation. Most learning algorithms do not perform well on data with a significant class imbalance. We therefore need to select the best method—e.g. oversampling—for addressing the class imbalance when constructing the machine learning model.

4. **Learning Algorithm Selection**: There are a number of well-known supervised machine learning algorithms (e.g. decision trees, support vector machines, naive bayes) which can be used to generate a classification model, however, it is not clear which will algorithm perform best for the service request data. We therefore need to select the best algorithm to use when constructing the machine learning model.

5. **Hyper-Parameter Optimization**: Each learning algorithm has a number of so-called hyper-parameters that control how the algorithm learns (e.g. the kernel function for the support vector machines algorithm, the number of hidden layers used in the neural network algorithm, and the number of neighbors in k-nearest neighbors algorithm). We therefore need to select the best value of each hyper-parameter associated with the selected learning algorithm.

The machine learning process can produce a wide range of different models depending on the decisions made by the knowledge engineer. For example, if the knowledge engineer decides to use naive bayes instead of support vector machines for the learning algorithm, the machine learning process will produce a distinctly different model Φ.

The overall objective for the knowledge engineer is to, given the wide range of models that can be generated, generate the specific model that performs best for the problem under consideration. We refer to this task as the model selection problem.

We distinguish between three distinct approaches to the model selection problem. These approaches are depicted in Figure 5.1. The first approach, which we refer to as "algorithm-down", is to focus on the learning algorithm and associated hyper-parameter (Issues 4, 5 above). The second approach, which we refer to as "algorithm-up", focuses on the algorithm and the data pre-processing related decisions (Issues 1-4 above). Finally, the third approach, "algorithm-up-down", combines the "algorithm-down" and "algorithm-up" approaches to address the end-to-end machine learning process (Issues 1-5).

In this work we focus on the "algorithm-up" approach to model selection. Based on our experiences in solving complex engineering problems, we believe that the external parameters are, in general, critical to overall effectiveness of the machine learning model. However, there is a distinct lack of existing work that addresses how the knowledge engineer can systematically address the problem of optimizing these external parameters. Furthermore, there are a number of well-known techniques, such as Bayesian optimization [Bergstra et al., 2011], that have already addressed the "algorithm-down" approach and can be readily used by knowledge engineers.

In the "algorithm-up" approach, we define a model parameter, θ_i, to be any aspect of the machine learning process that can be controlled by the knowledge engineer. The parameter space, denoted Θ, for a particular problem is the set of all model parameters $\theta_1, \theta_2, ..., \theta_n$. For example, in the escalation problem the model parameter space, Θ, could be defined as:

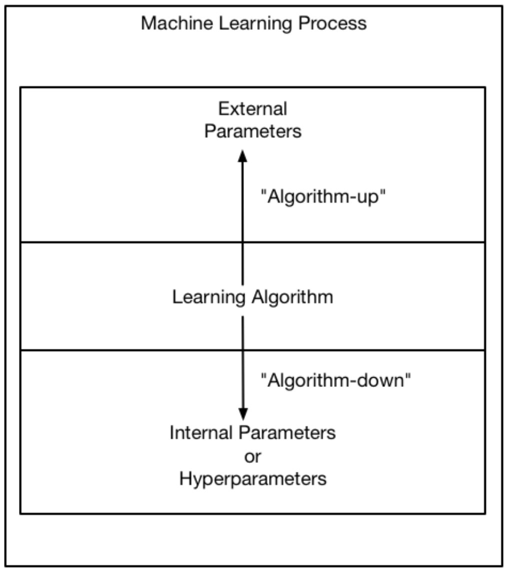


Figure 5.1: Layering the model selection problem into "algorithm-up" and "algorithm-down" approaches

$$\begin{aligned}\theta_1 &= \textit{Feature Selection Method}\\ \theta_2 &= \textit{Misclassification Cost Method}\\ \theta_3 &= \textit{Class Imbalance Method}\\ \theta_4 &= \textit{Learning Algorithm}\end{aligned}$$

Each parameter θ_i is associated with a set of m possible values, denoted $\alpha_{i,j}$ $(j = 1, 2, ..., m)$, that the parameter can take. For example, the range of values for the learning algorithm parameter, θ_4 in the example above, could be

$$\begin{aligned}\alpha_{4,1} &= \textit{Naive Bayes} \\ \alpha_{4,2} &= \textit{Support Vector Machines} \\ \alpha_{4,3} &= \textit{Random Forests}\end{aligned}$$

In order to formalize the model selection problem with external parameters, we have drawn upon the algorithm selection framework from earlier work ([Rice, 1976], [Smith-Miles, 2008]). Our adaptation of this framework consists of five major blocks shown in Figure 5.2. We describe each of the five blocks below.

The three major inputs to the machine learning process are represented by the following blocks in Figure 5.2: the problem space P, the parameter space Θ, and the performance metric space Γ. The problem space, denoted P, is the set of all data that has been collected for the problem of interest. Inside of this problem space, we will select a subset of data to serve as the training/test data-set, denoted D, for creating the machine learning model. Each instance in this training/test data-set, denoted $d \in D$, consists of a service request X and a prediction label y. The parameter space Θ is the set of all machine learning process parameters, $\theta_1, \theta_2, ..., \theta_n$, that can be controlled by the knowledge engineer. The performance metric space, denoted Γ, is the set of evaluation metrics—e.g. accuracy, f-measure, and g-mean—that we will use to evaluate the performance of the machine learning model.

The machine learning process itself is represented by the following two blocks in Figure 5.2: model creation and model evaluation. The model creation block takes the data-set D and selected model parameter values A as input and produce the machine learning model $\Phi(A; D)$. The model evaluation block takes a machine learning model $\Phi(A; D)$ and performance metric $\gamma \in \Gamma$ as input and produces the model evaluation $\gamma(\Phi(A; D))$ as output.

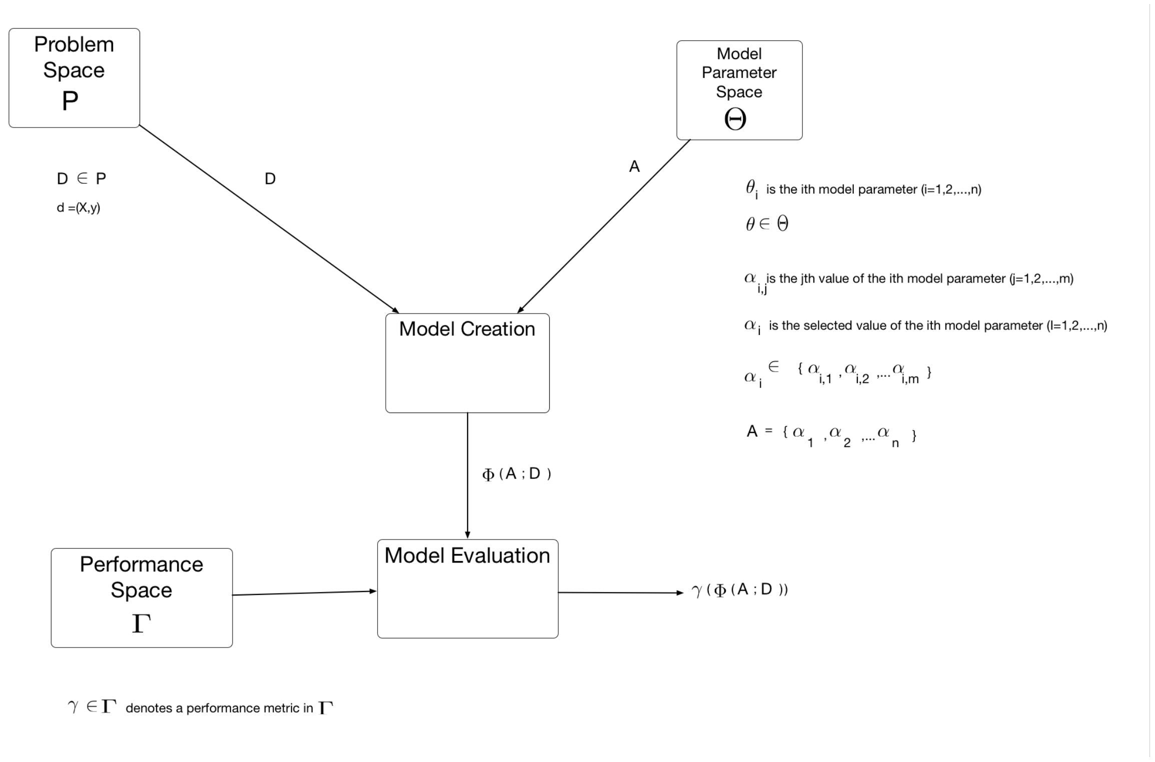

Problem Space P
Model Parameter Space Θ
D ∈ P
d =(X,y)
D
A
θ_i is the ith model parameter (i=1,2,...,n)
θ ∈ Θ
α_i,j is the jth value of the ith model parameter (j=1,2,...,m)
α_i is the selected value of the ith model parameter (I=1,2,...,n)
α_i ∈ { α_i,1 , α_i,2 ,... α_i,m }
A = { α_1 , α_2 ,... α_n }
Model Creation
Φ (A ; D)
Model Evaluation
Performance Space Γ
γ (Φ (A ; D))
γ ∈ Γ denotes a performance metric in Γ

Figure 5.2: Formulation of the machine learning model selection problem

Using Figure 5.2 the model selection problem, to be performed by the knowledge engineer, can now be formally described as follows:

1. Define the external parameters $\theta_1, \theta_2, ..., \theta_n$. For each parameter θ_i $(i = 1, 2, 3, ..., n)$, define the corresponding values $\alpha_{i,j}$ $(j = 1, 2, 3, ..., m)$. This is the model parameter space in Figure 5.2.

2. Select the set of training/test instances D from the problem space P that will be used create the machine learning model. This is the output from the problem space P in Figure 5.2.

3. For each parameter θ_i $(i = 1, 2, ..., n)$, select a value α_i $(i = 1, 2, 3, ..., n)$ where $\alpha_i \in \{\alpha_{i,1}, \alpha_{i,2}, ..., \alpha_{i,m}\}$. The resulting parameter vector, denoted as A, is

$$A \rightarrow \{\alpha_1, \alpha_2, ..., \alpha_n\}. \tag{5.2}$$

This is the output from the parameter space Θ in Figure 5.2.

4. In order to create the model, apply the machine learning process to the data-set of training/test examples D. The resulting model, Φ, is denoted as

$$\Phi(A; D) \ . \tag{5.3}$$

   This is the output from the model creation in Figure 5.2. It is important to note that in Equation 5.3 the parameter vector A is the variable of interest. We have included the data-set D in the model definition to emphasize the fact the training data-set is integral to resulting model.

5. Evaluate the machine learning model with respect to the performance metric $\gamma \in \Gamma$ for the problem under consideration.

$$\gamma(\Phi(A; D)) \tag{5.4}$$

   This is the output from the model evaluation in Figure 5.2.

6. Based on the model evaluation, determine the combination of learning algorithm and values for the external parameters that optimize the performance metric for the problem under consideration. Let A^* denote the parameter value vector that optimizes Equation 5.4. For the given set of training/test instances $D \in P$, find the parameter value vector A^* which optimizes $\gamma(\Phi(A^*; D))$.

5.3 Literature Survey

In section 5.2 we identified three different approaches to the machine learning model selection problem: "algorithm-up", "algorithm-down", and "algorithm-up-down". Existing work in the machine learning community has primarily addressed the "algorithm-down" approach to model selection which focuses on optimizing the learning algorithm and associated hyper-parameters. Inside the "algorithm-down" body of work, there are three major areas: selection of the best learning algorithm, optimization of the hyper-parameters associated with a selected algorithm, and combined algorithm selection and hyper-parameter optimization. The purpose of this section is to summarize the work in each of the areas related to "algorithm-down" optimization. We then discuss some of the challenges and limitations related to the "algorithm-down" approach for real-world knowledge engineering problems. This leads to the motivation for the "algorithm-up" approach which is then developed in Section 5.4.

The first major area of work addresses algorithm selection problem: which machine learning algorithm will produce the best model for my data-set. There are two general approaches that have been developed in the machine learning community for algorithm selection. The first approach, meta-learning, involves creating machine learning model that can predict of the performance of different learning algorithms given a particular data-set [Smith-Miles, 2008]. In order to create a machine learning model for predicting algorithm performance we first need a data-set of how different algorithms perform. To this end, the Statlog project [Michie et al., 1994] generated meta-data statistics such as number of features, class entropy, attributes and classes correlations for several data sets with the aim of comparing the performance of a fixed set of algorithms. The meta-data from the Stat-log project was then used to generate rules for when to apply particular algorithms ([Gama and Brazdil, 1995], [Lindner and Studer, 1999]). The METAL project [Brazdil et al., 2003] expanded on the earlier work in Statlog to address classification and regression problems. In particular, METAL introduced the use of simple and fast learners called landmarkers for predicting the performance of algorithms [Pfahringer et al., 2000]. To generate rules, both the landmarkers and the learning algorithms are first trained on a given set of data-sets. The performance of a landmarker on new data-sets is then used to rank algorithms based on their likelihood of performing well based on their similarity to the landmarker.

The second general approach to algorithm selection is to combine multiple machine learning models into an ensemble or collection of models. Within ensemble learning there are two well-known methods: bagging and boosting. Bagging [Breiman, 1996] uses multiple models, learned on random samples of the same data-set, in order to reduce the variance of the predictions. Boosting [Schapire, 1990] changes the data sampling distribution for each iteration so that the number of misclassified instances from the previous iteration are increased. This effectively enables each model to "correct" for the mistakes of the previous model and, in general, increases the predictive power of the ensemble as a whole. For both bagging and boosting, the ensemble performs a majority voting on new instances in order to determine the class prediction.

After the algorithm is selected, there are a number of algorithm-specific parameters that adjust how the algorithm operates. For example, when applying the artificial neural network learning algorithm we need to select the number of nodes in the hidden layer of the neural network. The second major area of work addresses the optimization of these so-called hyper-parameters for the problem under consideration. A large body of work ([Strijov and Weber, 2010] [Bergstra et al., 2011], [Bergstra et al., 2013]) has addressed tuning the hyper-parameter using Bayesian optimization techniques. The idea in Bayesian optimization is we assume a prior distribution for the objective function. We then perform experiments and sequentially refine this function as data are observed via Bayesian posterior updating.

The third major area of existing work addresses the combined algorithm selection and hyper-parameter optimization problem. Auto Weka [Thornton et al., 2013] treats the choice of the learning algorithm as an additional hyper-parameter in the model and then shows how the resulting hierarchical hyper-parameter optimization problem can be solved using Bayesian optimization methods such as Tree-structured Parzen Estimator [Bergstra et al., 2011] and Sequential Model-Based Algorithm Configuration [Bergstra et al., 2013]. Meta-collaborative filtering [Smith et al., 2014] attempts to find combinations of learning algorithms and associated hyper-parameters that will perform well for a given data-set based on the performance for past data-sets. This is similar to earlier work in landmarking [Pfahringer et al., 2000] however incorporates the hyper-parameters in addition to the learning algorithm.

In real-world applications the external machine learning process parameters have a significant impact on how well the model solves the problem under consideration and it is important to consider alternatives for all aspects of the process in addition to the learning algorithm and associated hyper-parameters. At the same time there is a lack of easy to implement methods for exploring alternatives which often leads to important parameters, such as how the training/test data-set is created, to be selected ad-hoc. In order to address this problem we need an easily applicable approach for optimizing the external parameters in the machine learning process approach that can be implemented in a wide range of technology/software environments.

5.4 Theory: Representation-Based Model for Machine Learning Model Selection

In this section we show the application of the Integrated Meta-Representational Model (IMRM) to the model selection problem formulated in Section 5.2. This leads to the reformulation of the *Internal* layer for the model selection problem as an experimental problem. We then show how the incorporation of a statistical design of experiments (DOE) method provides an efficient approach to exploring the experimental space of possible models. Lastly, we describe the overall process for translating a given machine learning problem into a design of experiments problem so that the DOE method can be applied.

5.4.1 Layering the Machine Learning Model Selection Problem

In this section we apply the Integrated Meta-Representational Model (IMRM), developed in Chapter 3, to structure the machine learning model selection problem into five layers so that the appropriate engineering domains and methods and tools can be applied. To this end, the section is organized into three parts corresponding to the three high-level steps involved in applying the IMRM. First, we determine subject-matter for each of the five representational layers. Second, based on the subject-matter to be represented at each layer, we identify the relevant engineering domains and select the necessary methods and tools from these domains. Third, we integrate the selected methods and tools into an comprehensive process methodology.

The *External* layer subject-matter is the inputs to the machine learning process. Following from Figure 5.2 there are three main inputs to the machine learning process: the data-set, D, that provides the problem instances $D \in P$, the machine learning parameters $\theta_1, \theta_2, ..., \theta_n$ and associated values $\alpha_{i,1}, \alpha_{i,2}, ..., \alpha_{i,m}$, and the performance metric $\gamma \in \Gamma$. The *Outer* layer of the model selection problem, which satisfies the problem statement, is the machine learning model $\Phi(A^*; D)$ that optimizes the performance metric γ.

The *Internal* layer subject-matter is the optimal machine learning model $\Phi(A^*; D)$. The range of possible models for a particular machine learning process can be represented as an n by m matrix shown in Table 5.1 where matrix rows θ_i $(i = 1, 2, ..., n)$ are the set parameters for creating a machine learning model and the matrix columns $\alpha_{i,j}$ $(j = 1, 2, ..., m)$ are the possible value for factor θ_i. In order to create a model the knowledge engineer selects one column value $\alpha_i \in \{\alpha_{i,1},\ \alpha_{i,2},\ ...\ \alpha_{i,m}\}$ for each of the matrix rows θ_i $(i = 1, 2, ..., n)$.

Determining the optimal combination of values, denoted by the parameter vector A^*, requires building and testing different models. Each one of these models can be

Value / Parameter	Value 1	Value 2	$\cdots$	Value m
θ_1	$\alpha_{1,1}$	$\alpha_{1,2}$	$\cdots$	$\alpha_{1,m}$
θ_2	$\alpha_{2,1}$	$\alpha_{2,2}$	$\cdots$	$\alpha_{2,m}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
θ_n	$\alpha_{n,1}$	$\alpha_{n,2}$	$\cdots$	$\alpha_{n,m}$

Table 5.1: Experimental space for the machine learning model selection problem

thought of as an experiment to be performed. For most real engineering applications, the number of experiments necessary to exhaustively explore Table 5.1 and determine the optimal parameter vector $A^* = \{\alpha_1^*, \alpha_2^*, ..., \alpha_n^*\}$, is prohibitively large. For example, exhaustively building and testing 5 parameters with 5 possible values each would require 3,125 experiments (5^5) to be performed by the knowledge engineer. We now reformulate the model selection problem as the following design of experiments (DOE) problem: what is the minimal set of experiments that need to be performed in order to determine optimal parameter vector A^*?

Table 5.2 shows the layers, representational subject matters, domains, and methods for the service level escalation problem. At the *External* layer we need to model the domain knowledge of the model selection problem under consideration. In the context of engineering applications, predictive models are often developed in the context of an engineering process. To this end we draw upon three models from the CommonKADS methodology [Schreiber, 1994]—the organization model, the agent, and the task model—to capture and represent the technical, organizational, and financial considerations that are relevant to the predictive model.

A considerable body of literature exists in the development of methods for performing experiments ([Fisher, 1935], [Taguchi and Konishi, 1987], [Fowlkes and Creveling, 1995], [Srinagesh, 2006]). Inside of the planned Design of Experiments (DOE) methods, we draw upon the Taguchi method [Taguchi and Konishi, 1987] to address the problem of planning the experiments to determine the optimal parameter vector A^* for the machine learning model. The Taguchi method was developed to optimize a quality characteristic of a process or product, e.g. the number of defects for a semiconductor manufacturing process, and thus is particularly well suited to the task of optimizing a performance metric for a machine learning model. The application of the Taguchi method maps to the *Outside-In*, *Internal*, *Inside-Out*, and *Outer* layers.

For any non-trivial set of experiments we need to automate the process of building and testing the machine learning models at the *Inside-Out* layer. For small to medium data-sets with less than one million data instances there are two primary options for automation: open-source tools (Weka [Witten and Frank, 2005] or R [R Development Core Team, 2008]) or commercial data-mining tools (SAS or Matlab). For larger data-sets with millions of instances, it is necessary to use big-data software tools (e.g. Hadoop) to create the machine learning models in a timely manner.

In this work we have selected the Weka machine learning work-bench [Witten and Frank, 2005] as the software tool for implementing the planned experiments at the *Inside-Out* layer. Weka was selected because it supports a wide range of machine learning algorithms and has an interactive interface that makes it easy to quickly build-and-test different models. Lastly, at the *Outer* layer we analyze the results from the experiments and use the optimal parameters to generate a prediction model. Analysis of the Means (ANOM) [Phadke, 1989] is used to create an experiment-based model of the factor effects. Weka is used to build the model for the verification experiment and final prediction model.

Figure 5.3 shows the representation-based model resulting from the integration of the methods and tools described above. The model is implemented by sequentially stepping through each layer as follows:

1. ***External* layer**: Capture the work-process that the machine learning model will

Layer	Subject Matter	Domains and Representational Methods/Tools		
		Knowledge Engineering	Machine Learning	Robust Design
External	Organizational context and current work process for service request escalation	CommonKADS Organization, Agent, and Task Models		
Outside-In	Performance metric			Taguchi method: Signal-to-Noise Ratio
Internal	Experimental Design			Taguchi method: Orthogonal Array for Planned Experiments
Inside-Out	Perform Experiments		Weka Machine Learning Workbench	
Outer	Model Selection			Taguchi method: Analysis of the Means (ANOM)

Table 5.2: Layer-subject matter-representational tools/methods matrix for predicting service request escalation

automate by creating a CommonKADS Agent/Task Model.

2. ***Outside-In* layer**: Determine the performance metric that the machine learning model needs to optimize. Transform the performance metric into a signal-to-noise (s/n) ratio function to be maximized.

3. ***Internal* layer**: Determine the control, signal, and noise factors for the machine learning model. Select the appropriate orthogonal array and assign the factors and the factor settings or levels, respectively, to the columns and rows of the orthogonal array matrix.

4. ***Inside-Out* layer**: Perform the experiments in the Weka machine learning workbench. Record the prediction results for the test data-set and compute the s/n ratio for each experiment.

5. ***Outer* layer**: Perform Analysis of the Means (ANOM) to compute the s/n ratios for each combination of factor and level setting. Select the combination of factor

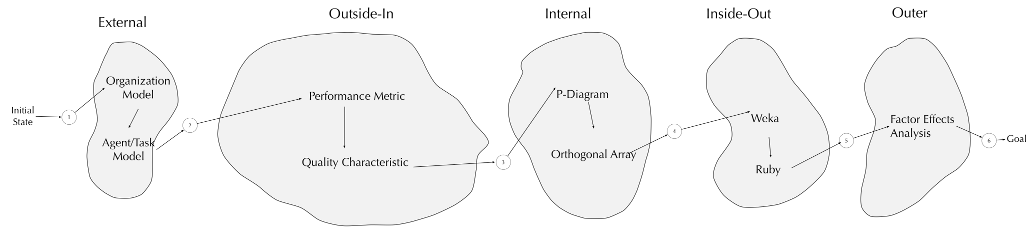


Figure 5.3: Representation-based model for predicting service request escalation

settings that maximize the overall s/n ratio. Perform a verification experiment to check the results.

The implementation of the representation-based model is described in Section 5.5.

5.4.2 Design of Experiments for Machine Learning Model Selection

In this section we show the mapping between the Taguchi Design of Experiments (DOE) methods and the external model selection problem. To this end, we first transform the model selection problem from the machine learning domain to the design of experiments domain. We then show how the optimal level settings are determined for the experimental factor levels using Analysis of Means (ANOM). Lastly, we show the transformation of the optimal level settings back to the machine learning problem in order to determine the optimal values for the external parameter vector.

Before we describe the Taguchi method, we first define the basic terminology and notation used in experimental design. The variables or inputs of interest in any experiment are called factors. In a given experiment the value or setting of a given factor is called the level of that factor. Let f_i $(i = 1, 2, ..., n)$ denote the ith factor in the factor space of the experiment, and let $L_{i,j}$ $(j = 1, 2, ..., m)$ denote the jth level of the factor i. For the purposes of this work we make the simplifying assumption that each factor contains m levels.

An experiment consists of a user-selected level setting for each one of the n factors. Let $L_i^k \in \{L_{i,1}, L_{i,2}, ..., L_{i,m}\}$ denote the selected level setting for the ith factor in the kth experiment. The kth experiment, denoted E^k, is therefore defined as follows

$$E^k \equiv \{L_1^k, L_2^k, ..., L_n^k\} \tag{5.5}$$

In the Taguchi method we are trying to optimize a quality characteristic. The factors that influence a quality characteristic of a system or process of interest can be organized into three classes:

1. **Signal Factors**: These parameters are set by the user or operator of the system to obtain the intended value for the response of the process.
2. **Noise Factors**: The noise factors are the input parameters to the system, which, in general are uncontrollable.
3. **Control Factors**: The control factors are the parameters of the system controlled by the designer/developer in order to optimize the quality.

The control factors f_i $(i = 1, 2, ..., n)$ are the parameters that can be adjusted to optimize the quality characteristic. The signal factors are the output of the system or process that we are measuring using the quality characteristic.

In the Taguchi method, we define the s/n ratio, denoted as η, as a function of the quality characteristic that we then desire to maximize. The most basic but inefficient way to maximize η is to try exhaustively, every possible combination of level settings for each factor an then determine the combination of level settings for which η is maximized. This approach is called a full factorial design and requires m^n experiments where n is the number of factors and m is the number of levels for each factor (for the purposes of explanation we assume each factor has the same number of levels).

Experiment	Factor 1 (f_1)	Factor 2 (f_2)	Factor 3 f_3	Factor 4 f_4)
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Table 5.3: Orthogonal array for a four factor - three Level experimental design

In contrast to the full factorial approach described above, the Taguchi method uses special matrices, called orthogonal arrays [Taguchi and Konishi, 1987], to plan the experiments to be performed, and then, Analysis of the Means [Phadke, 1989] to efficiently determine the optimal level settings for each factor. Table 5.3 shows an example of an orthogonal array for the case of four factors f_i $(i = 1, 2, 3, 4)$, where each factor has three levels $(i = 1, 2, 3)$. The full factorial approach for this case would require $3^4 = 81$ experiments. In contrast, as seen in Table 5.3, the Taguchi method requires only 9 experiments.

Using Table 5.3 we can illustrate the main points of how orthogonal arrays are used in the Taguchi method:

1. Each row of the orthogonal array corresponds to an experiment. For example, the fourth row, i.e. experiment 4, corresponds to an experiment with the level settings 2, 1, 2, and 3, for factors 1, 2, 3, and 4, respectively. In the notation of Equation 5.5; $E^4 = (L_{1,2}, L_{2,1}, L_{3,2}, L_{4,3})$.

2. The total number of experiments that need to be performed is equal to the number of rows in the orthogonal array. For the case of Table 5.3 where we have 4 factors with 3 levels for each factor, 9 experiments are required in order to find the optimal level settings.

3. For any two columns in the orthogonal array every ordered pair of level settings occurs in exactly one row. For example, in Table 5.3 there are no duplicate pairs of level settings for factors f_1 and f_2. As a result,the columns of the orthogonal array columns are independent to each other and each level settings occurs the same number of times across the columns. These two properties enable the factors to be tested pair-wise as opposed to one factor at a time which results in significant efficiency gains with respect to the one-factor-at-a-time and factorial designs.

4. The advantage of the Taguchi method is that it enables the experimentalist to span the experimental space with a minimal set of experiments, 9, in the present case compared to the 81 experiments require in the full factorial approach.

There are two general ways to apply the Taguchi method [Phadke, 1989]. In the first approach, all the control factor are assumed to be independent, and as a consequence, their effect on the signal factor is additive. In this case, orthogonal arrays [Taguchi and Konishi, 1987], such as the one shown in Table 5.3, can be directly used to plan the experiments and analyze the results. For the case of n factors and m levels per a factor, it can be shown that [Phadke, 1989] the number of experiments, g is given by

$$g = 1 + n(m - 1). \tag{5.6}$$

In the second approach, the factors are dependent on each other. In this case, the dependencies between the factors must first be modeled using linear graphs [Taguchi and Konishi, 1987]. We then assign each one of these factor interactions, modeled in the linear graph, to a separate column of the orthogonal array. Performing the experiments in the orthogonal array then allows us study interactions between the factors while also measuring the effects of the individual factors. A detailed discussion of linear graphs and how to study interaction between the factors can be found in [Phadke, 1989].

In this work, for simplicity and for the purposes of illustrating the use of the Taguchi method to the machine learning model selection problem, we assume no interaction between the different parameters in the machine learning process. The basic Taguchi method, based on the assumption of independence, essentially comprises of three distinct stages [Phadke, 1989] summarized as follows:

1. Plan the experimental design using orthogonal arrays.

   (a) Determine, n the number of control factors we want to test, e.g., in Table 5.3 $n = 4$.

   (b) Determine m, the number of levels for each factor, e.g., in Table 5.3 $m = 3$.

   (c) Determine the orthogonal array for the combination of n control factors and m levels per factor. A catalog of orthogonal arrays for the common values of n and m can be found in [Taguchi and Konishi, 1987], [Phadke, 1989]. The orthogonal array for the case of $n = 4$, $m = 3$, shown in Table 5.3, specifies the number of experiments required (9 in this case) as well as the levels for each factor in a given experiment.

2. Perform the experiments as specified by the orthogonal array.

   (a) Determine the s/n ratio, η, to be measured in the experiments. The three different general forms of the s/n ratio function depending on the objective function for the quality characteristic:smaller-the-better, larger-the-better, and nominal-the-best [Phadke, 1989]. In the service request escalation example we use the s/n ratio function given by Equation 5.9.

   (b) For each experiment E^k, corresponding to one row in the orthogonal array, measure the corresponding value of η^k, the s/n ratio for the experiment k.

3. Perform an Analysis of the Means (ANOM) for the experimental data.

(a) Compute $\bar{\eta}$, the overall mean of the s/n ratio η across all experiments, as given by Equation 5.10.

(b) Compute the mean s/n ratio, denoted $\bar{\eta}(L_{i,j})$, for each factor level $L_{i,j}$ $(i = 1, 2, ..., n)(j =, 1, 2..., m)$ as given by Equation 5.11.

(c) Determine the combination of level settings for each control factor that maximize η as given by Equation 5.12, and then compute the predicted value of the optimal η as given by Equation 5.13.

(d) Perform an actual verification experiment to check the results obtained in Step (c).

We now develop the formal notation and analytical details for the process described above. In Step 1 of the Taguchi process we define the control factors and the levels associated with each control factor. Each machine learning model parameter, θ_i $(i = 1, 2, ..., n)$, will be treated as a factor, denoted as f_i, in an experiment. We denote this equivalence as follows:

$$\theta_i \equiv f_i \ (i = 1, 2, ..., n) \tag{5.7}$$

Each possible value that the model parameter θ_i can take, denoted $\alpha_{i,j}$ $(j = 1, 2, ..., m)$ in Equation 5.8, is then equivalent to a level, denoted as $L_{i,j}$, associated with factor f_i $(i = 1, 2, ..., n)$:

$$\alpha_{i,j} \equiv L_{i,j} \ (i = 1, 2, ..., n), (j = 1, 2, ..., m) \tag{5.8}$$

In Step 2 of the Taguchi method we define the s/n ratio to be measured. For the model selection problem we are trying to optimize a performance metric $\gamma \in \Gamma$ (Section 5.2). In our treatment, this performance metric γ is now equivalent to the quality characteristic in the Taguchi method. Each experiment generally involves several measurements of the quality characteristic. Let s denote the number of measurements in each experiment and γ_i^k $(i =, 1, 2, ..., s)$ denote the ith measurement of the quality characteristic for the kth experiment

In the case of the service request escalation problem, quality characteristic γ_i^k $(i = , 1, 2, ..., s)$, representing the escalation, needs to be minimized. Minimizing γ_i^k is equivalent to maximizing the following s/n ratio [Phadke, 1989]:

$$\eta^k = -10log_{10}\frac{1}{s}\sum_{i=1}^{s}(\gamma_i^k)^2 db, \tag{5.9}$$

where η^k is the mean square of the performance metric γ^k expressed in decibels.

In Step 3 of the Taguchi process, Analysis of the Means (ANOM) [Fowlkes and Creveling, 1995] is used to determine the effect of each factor on the s/n ratio η^k and determine the optimal level settings. The ANOM technique begins with computing the overall experimental mean of the s/n ratio across all experiments. Let $\bar{\eta}$ denote the overall experimental mean. For the case of the basic Taguchi method with n independent control factors, each with m levels, $\bar{\eta}$ is given by

$$\bar{\eta} = \frac{1}{1 + n(m-1)} \sum_{i=1}^{1+n(m-1)} \eta^k db, \tag{5.10}$$

where $1 + n(m - 1)$ is the total number of experiments to be performed.

Next, for each level setting $L_{i,j}$ we compute the experimental mean of all experiments which use ths factor setting, i.e. where $L_{i,j} \in E^k$. Let $\bar{\eta}(L_{i,j})$ denote the mean of the s/n ratios for all experiments containing level $L_{i,j}$, $\bar{\eta}(L_{i,j})$ is given by

$$\bar{\eta}(L_{i,j}) = \frac{\sum\{\eta^k | L_{i,j} \in E^k \mid}{|\eta^k | L_{i,j} \in E^k|}, \tag{5.11}$$

where $|L_{i,j} \in E^k \mid$ is the number of number of experiments which use factor setting $L_{i,j}$.

Each factor f_i $(i = 1, 2, ..., n)$ has an optimal level setting, denoted as $\hat{L}_i$, that maximizes the s/n ratio η given by

$$(\hat{L}_i | \hat{L}_i \in L_i \wedge \bar{\eta}(\hat{L}_i) = max(\bar{\eta}(L_i))). \tag{5.12}$$

Since the basic Taguchi method assumes that the experimental factors are independent, then

$$\bar{\eta}^k = \bar{\eta} + \sum_{i=1}^{n} \bar{\eta}(L_{i,j}) | L_{i,j} \in E^k db \tag{5.13}$$

and the experimental optimal, E^*, is the combination of the optimal level for each factor

$$E^* = L^* = \{L_1^*, L_2^*, ..., L_n^*\} \tag{5.14}$$

The last step is transforming the experimental optimal, E^*, into the optimal external parameters for the machine learning process. In order to translate the optimal experiment back to the machine learning domain, we first map each optimal level setting L_i^* $(i = 1, 2, ..., n)$ to a corresponding optimal value in the external parameter space α_i^*

$(i = 1, 2, ..., n)$ for the machine learning model (Section 5.2).

$$L_i \equiv \alpha_i^* \tag{5.15}$$

The parameter vector, A^*, for the machine learning model that optimizes our objective, $y(\Phi(A; D))$, is then given by:

$$A^* = \{\alpha_i^*, \alpha_2^*, ..., \alpha_n^*\}. \tag{5.16}$$

5.5 Application: Process Methodology for Predicting Service Request Escalation

In this section we demonstrate the implementation of the representation-based model shown in Figure 5.3 using the problem to predict whether or not a particular customer support case will need to be escalated in priority in order to be resolved in a timely manner. This problem was described earlier in Section 5.2

The treatment of the process methodology is organized into five parts—*External*, *Outside-In*, *Internal*, *Inside-Out*, and *Outer*–corresponding to five layers of the Integrated Meta-Representational Model. For each layer, we first identify the overall inputs and outputs and the layer's role in creating a solution to the overall problem. We then describe the methods and techniques used at the layer to transform the inputs into outputs. Lastly, we walk through the implementation of the layer for the service request escalation problem under consideration.

The networking company collected a data-set of 146,446 historical service requests to use in order to use for learning the predictive model. Each historical service request, denoted d_i $(i = 1, 2, ..., 146, 446)$, consists of a feature vector X_i and a label $y_i \in \{-1, +1\}$

indicating whether the service request was escalated (+1) or not (−1).

$$d_i = (X_i, y_i) \tag{5.17}$$

The learning task is to create the predictive model, denoted Φ, that for a given feature vector X will determine the prediction label y.

$$\Phi : X \to y \tag{5.18}$$

5.5.1 External Layer: Domain Knowledge Modeling

The *External* layer of the representation-based model, shown in Figure 5.3, addresses the collection of the relevant domain knowledge about the model selection problem under consideration. The representation of this domain knowledge is based on three models from the CommonKADS methodology [Schreiber, 1994]. The Organization model is used to represent the key organizational factors related to the decision process that the classification model will automate. The Agent and Task models are used to represent the work process and the structure of the individual tasks involved in decision process. Together, these models capture these factors in a structured format that can be integrated into the machine learning process in order to achieve our objective of optimizing the model with respect to some performance criteria $\gamma \in \Gamma$.

The *External* layer process for applying the Organization and Agent/Task models is as follows:

1. Organize the collected information using a **CommonKADS Organization model** [Schreiber, 1994]. The process for creating the Organization model is as follows: a) Extract the key organizational components—context, people, processes,

resources—from the information collected in Step 1; b) Create a diagram of the relationships between these entities; c) Identify problems or challenges in the current processes.

2. Interview network engineers about the currently manual work process for determining if a particular service request should be escalated. Techniques for work process interviewing are discussed in [Schreiber, 1994].

3. Construct an **Agent/Task model** [Schreiber, 1994] for the current work process. The process for creating the Agent/Task model is as follows: a) Create an agent for each person or resource involved in the current work process; b) Create a task for each step in the work process captured in Step 3; c) Relate agents to the tasks that they perform. Label tasks performed by a single actor as $<<$ *includes* $>>$ and tasks performed by multiple actors as $<<$ *uses* $>>$.

4. Interview stakeholders about the high-level organization factors involved in escalation. Important information to collect includes current work processes, the people and resources involved in these work processes, and cost metrics for the escalation.

5.5.2 Outside-In Layer: Performance Metric and Signal-to-Noise Ratio for Model Selection

The *Outside-In* layer of the representation-based model, shown in Figure 5.3, addresses the translation of the service request costs from the *External* layer into a specific performance metric for the machine learning model. The performance metric is then translated into a corresponding s/n ratio that we are trying to maximize.

The process for identifying the appropriate performance metric and defining the corresponding s/n ratio is as follows:

1. Define the performance metric based on the Organization and Agent/Task model from the External layer.

2. Identify the quality characteristic to be observed for the problem under consideration. The selected quality characteristic should be easy to measure, continuous, and monotonic. [Phadke, 1989]

3. Define the s/n ratio function to be optimized. The three main types of objective functions for the quality characteristic are: smaller the better, nominal is the best, and larger the better. A discussion of these objective functions can be found in [Phadke, 1989].

The overall objective for the predictive model is to minimize the additional escalation-related costs associated with resolving a set of service requests. In order to quantify this objective we worked with domain experts to determine the following costs:

1. A typical service request costs approximately \$250 to resolve.

2. Service requests that need to be escalated are considerably more expensive and generally cost \$250,000 to resolve because multiple support teams are involved and the hardware replacement is often used to solve the problem in a timely manner.

3. Proactive escalation of a service request, before the customer requests an escalation, will typically reduce the cost to \$50,000 per a case.

Table 5.4 summarizes the additional escalation costs associated with each outcome. The additional cost of a correct prediction (TP or TN) is \$0. If we incorrectly escalate a case that did not need to be escalated (FP) it costs \$49,750 over the base-line (\$50,000 - \$250). Likewise, a missed escalation (FN) will incur an additional cost of \$249,750 over the baseline \$250 (\$250,000 - \$250).

	Escalated	Not Escalated
Needed to be Escalated	\$0.00 (TP)	\$249,750 (FN)
Did not need to be Escalated	\$49,750 (FP)	\$0.00 (TN)

Table 5.4: Estimated costs for escalating a customer service request

Let $\hat{y}_i$ denote the predicted label for ith service request from the model Φ. There are four possible cases for the prediction: true positive, true negative, false positive, and false negative. A true positive (TP) is when the predicted label is escalate ($\hat{y}_i = +1$) and the actual service request label is also escalate ($y_i = +1$). Likewise, a true negative (TN) is when both $\hat{y}_i = -1$ and $y_i = -1$. Likewise, a false positive (FP) is when $\hat{y}_i = +1$ but ($y_i = -1$) and a false negative (FN) is $\hat{y}_i = -1$ but $y_i = +1$.

Let γ_i denote the additional escalation cost of ith service request incurred given by Table 5.4.

$$\gamma_i = \begin{cases} \$0 \ \textit{if TP}, \\ \$0 \ \textit{if TN}, \\ \$249,750 \ \textit{if FN}, \\ \$49,750 \ \textit{if FP}, \end{cases} \tag{5.19}$$

We want to minimize the total cost, denoted C, of the escalations over s service requests.

$$C = \sum_{i=1}^{s} \gamma_i \tag{5.20}$$

Minimizing Equation 5.20 is equivalent to maximizing the following s/n ratio given by Equation 5.21.

$$\eta = -10log_{10}[\frac{1}{s}\sum_{i=1}^{s}\gamma_i^2] \tag{5.21}$$

5.5.3 Internal Layer: Design of Experiments

The *Internal* layer of the representation-based model, shown in Figure 5.3, addresses the task of designing a set of experiments that can be used to systematically compare different parameter vectors for the machine learning model. The design of experiments is performed using the Taguchi method ()[Taguchi and Konishi, 1987], [Phadke, 1989]) in order to minimize the number of experiments necessary to fully explore the design space for the prediction model.

The process for applying the Taguchi method to design the experiments for the service request escalation problem is as follows:

1. Identify the control, signal, and noise factors for the experiments. The control factors are the factors that the knowledge engineer can control, in our case the learning parameters that we are using to optimize the machine learning model. The signal factors are the inputs to the experiment, in our case the historical customer service requests. The noise factors are any elements of the experiment that cannot be controlled by the knowledge engineer, e.g. the experience level of the technical support engineer assigned to the case.

2. Determine the different possible levels or parameter settings for each control factor. Potential control factors for the machine learning process include: the subset of features used; the sampling scheme used to create the training/test data-set; and the machine learning algorithm used to create the machine learning model.

3. Select the appropriate orthogonal array from the number of control factors, and

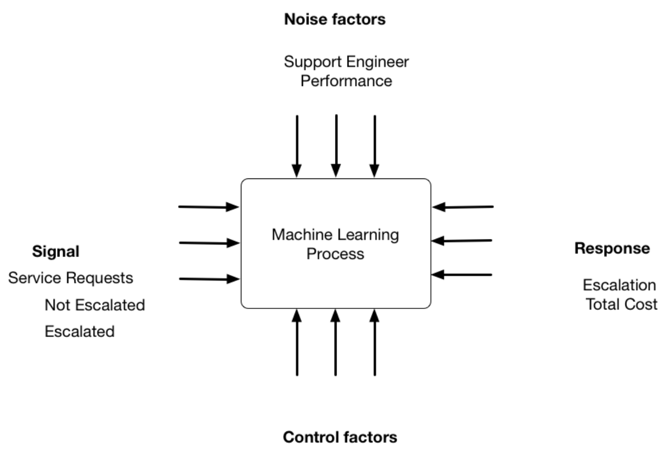


Figure 5.4: Experimental factors for predicting service request escalation

the number of levels for each control factor. Assign each factor to a column in the orthogonal array. Assign the factor levels to the array rows.

The *Internal* layer for the service request escalation problem resulted in the experimental design shown in Table 5.6. In order to create this experimental design we first modeled the service request escalation process is shown in Figure 5.4. The input signal to this process is the set of service requests for which we need to determine an escalation or no escalation label. Based on these labels there is a total escalation cost associated with the service requests. The two parameters that influence the accuracy of the labeling, and therefore the total escalation cost, can be divided into two classes: noise factors and control factors.

The noise factors for the learning process are the different workloads and levels of experience for the support teams. For example, if a service request that would not normally need to be escalated was assigned to an overloaded support team it could have resulted in an escalation because the team did not respond the customer promptly. Likewise, an exceptional support team could potentially avoid an escalation for a service request that typically would have required an escalation. Since these factors are external

Factor / Level	1	2	3
A. Machine Learning Algorithm	Naive Bayes	Random Forests	SVM
B. Classification Structure	Cascade (Domain Knowledge)	Cascade (Boosting)	Flat (Binary)
C. Feature Selection	Manual (Domain Knowledge)	Correlation (Chi-Squared)	Information Gain
D. Class Imbalance	Oversampling	Undersampling	Cost Sensitive Learning

Table 5.5: Machine learning process parameters for predicting service request escalation

to the labeling they are noise factors for our problem.

Table 5.5 shows the set of control factors and parameter settings for the service request escalation problem. There are four different control factors corresponding to the four major steps in the machine learning process. The feature selection factor addresses the method to be used to select subset features or attributes for the training/test data-sets. The class imbalance factor captures the method used to address the unbalanced distribution of positive and negative labels in the service request data-set. The learning algorithm factor addresses the learning method to be used for generating the machine learning model. Lastly, the classification structure factor addresses the model used to generate the final service request labeling, e.g. single decision or a cascade of several simpler decisions. The level settings for each of the four control factors, are shown in Table 5.5 and discussed below.

We selected three well-known machine learning algorithms for the levels of the machine learning control factor: naive bayes, support vector machines, and Random Forests ([Witten and Frank, 2005], [Hastie et al., 2009]). The naive bayes algorithm creates a simple probabilistic classification function based on applying Bayes theorem with strong (naive) independence assumptions. The support vector machines algorithm finds the separating hyperplane between the two classes in the data-set. Random forests is an ensemble learning method that creates multiple decision trees at training time, and then combines the separate prediction results from individual decision trees.

The classification structure factor controls the application of the learning algorithms to create the predictive model. The three level settings are: flat, boosting, and multi-stage. The flat setting is a simple binary one-level classifier for classifying an incoming service request. Boosting, which was discussed in Section 5.3, uses an ensemble of classifiers to predict the final label. The third option, multi-stage, attempts to replicate the existing manual process that engineers use to decide if a service request should be escalated.

The multi-stage classification structure, shown in Figure 5.5, is implemented as a cascade or series of machine learning models. The cascade processes makes three decisions for each service request. First, is the problem described in the service request a severe problem? Second, is the problem for a new or important product? Third, is the problem from an important customer? Each of these decision will produce one of three outcomes: no review, possible review candidate, or review. Any single review decision or any two review candidate decisions will cause the service request to be labeled "escalate", otherwise, the service request will be labeled "do not escalate".

The next step is to select the corresponding orthogonal array for planning the experiments. Table 5.5 shows that we have four factor, each with three levels. Therefore, if we assume that the control factors are independent, then for the case of the four factors we need to perform 9 $(1 + 4(3 - 1))$ experiments to determine the optimal level setting for each of the four factors. It is important to note that a factorial experimental design for testing these factors would require 81 (3^4) experiments while the Taguchi method only requires 9 experiments.

The corresponding L_9 orthogonal matrix for organizing these nine experiments was selected from [Phadke, 1989] and is shown below in Table 5.6. Each row in Table 5.6 describes one experiment that will need to be performed. For example, in experiment

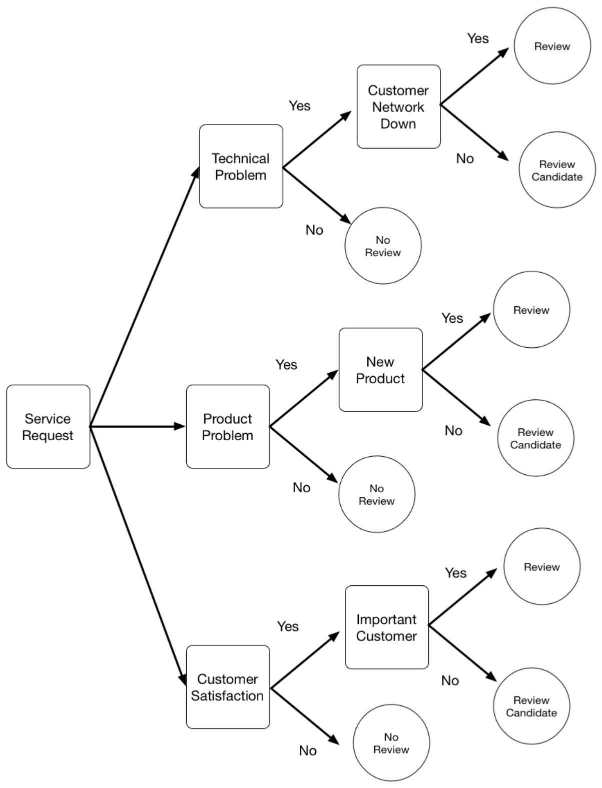


Figure 5.5: Multi-stage classifier cascade for predicting service request escalation

Experiment	Machine Learning Algorithm (f_1)	Feature Selection (f_2)	Classification Structure (f_3)	Class Imbalance (f_4)
E^1	Naive Bayes	Domain Knowledge	Cascade (Domain Knowledge)	Oversampling
E^2	Naive Bayes	Correlation	Cascade (Boosting)	Undersampling
E^3	Naive Bayes	Information Gain	Flat (Binary)	Cost Sensitive Learning
E^4	SVM	Domain Knowledge	Cascade (Boosting)	Cost Sensitive Learning
E^5	SVM	Correlation	Flat (Binary)	Oversampling
E^6	SVM	Information Gain	Cascade (Domain Knowledge)	Undersampling
E^7	Random Forest	Domain Knowledge	Flat (Binary)	Undersampling
E^8	Random Forest	Correlation	Cascade (Domain Knowledge)	Cost Sensitive Learning
E^9	Random Forest	Information Gain	Cascade (Boosting)	Oversampling
E^*	Random Forest	Information Gain	Domain Knowledge	Cost Sensitive Learning

Table 5.6: Orthogonal array of the nine experiments for the service request escalation prediction model

3 (E^3) the classification function, Φ is created using a naive bayes learning algorithm with features selected using based on information gain, and the misclassification costs are weighted according the to the escalation costs.

5.5.4 Inside-Out Layer: Experimentation

The *Inside-Out* layer of the representation-based model, shown in Figure 5.3, addresses the implementation of the planned experiments from the *Internal* layer.

The process for implementing the experiments is as follows:

1. Perform each experiment by constructing the machine learning model corresponding to the parameters specified in the experimental design.

Experiment	TP	TN	FP	FN	S/N (db)
E^1	130894	521	14575	456	-125.619
E^2	124397	672	21072	305	-128.768
E^3	130780	557	14689	420	-125.675
E^4	120055	766	25414	211	-130.385
E^5	11550	759	29919	218	-131.801
E^6	124812	707	20657	270	-128.592
E^7	136821	956	8648	21	-121.017
E^8	137508	961	7961	16	-120.298
E^9	138385	963	7084	14	-119.284
E^*	140608	899	4861	78	-116.031

Table 5.7: Evaluation of the service request prediction models with respect to total cost

2. Apply the machine learning model to the test data-set and record the prediction decision, escalate or do not escalate, for each service request. Record the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each machine learning model.

3. Compute the corresponding s/n ratio for each experiment by computing Equation 5.21 from the *Outside-In* layer using the TP, TN, FP, FN counts from Step 2.

For the service request escalation problem the nine experiments shown in Table 5.6 were implemented using knowledge flows in the Weka machine learning workbench [Witten and Frank, 2005]. Each knowledge flow consisted of four sequential steps: training/test data-set, feature selection, model learning, and evaluation. The predictions from Weka were then processed using a series of scripts, written in Ruby, in order to compute the s/n ratios and total costs. Table 5.7 shows the counts for true positives (TP), true negatives (TN), false positives (FP), false negatives (FN), and the resulting s/n ratio for each of the nine experiments. The experimental results will be discussed in detail in Section 5.6.

Factor	Factor Level	Factor Effect (db)
Machine Learning Algorithm	Naive Bayes	-1.29
Machine Learning Algorithm	SVM	-4.86
Machine Learning Algorithm	Random Forests	5.2
Feature Selection	Domain Knowledge	-0.27
Feature Selection	Correlation	-1.55
Feature Selection	Information Gain	0.884
Classification Structure	Cascade (DM)	0.56
Classification Structure	Cascade (Boosting)	-0.74
Classification Structure	Flat	-0.76
Class Imbalance	Oversampling	-0.167
Class Imbalance	Undersampling	-0.72
Class Imbalance	Cost Sensitive	-0.05

Table 5.8: Factor effects for the machine learning model

5.5.5 Outer Layer: Optimization of the Model Parameters

The *Outer* layer of the representation-based model, shown in Figure 5.3, addresses the analysis of the experimental results from the *Inside-Out* layer in order to determine the factor settings that maximize the s/n ratio. The Analysis of the Means (ANOM) process for analyzing the experimental results is described in detail in Section 5.4.2. This section focuses on the experimental results of applying this process to the service request escalation problem.

The experimental results for the service request escalation problem from the *Inside-Out* layer consists of the s/n ratios computed for each of the nine experiments shown in 5.7. The signal to noise ratio for each factor level was calculated as the average s/n ratio of all the experiments that contained that factor level (Equation 5.11). For example, the naive bayes algorithm is a factor in experiments 1, 2, 3 and, therefore, the s/n value for the naive bayes setting is the average of experiments 1, 2, and 3 (see Table 5.6). Table 5.8 shows the s/n ratios for each of the three level settings for the four control factors.

We now select the optimum level for each factor in order to maximize the s/n ratio (Equation 5.14). From the results of Table 5.8, the combination that yields the maximum s/n ratio is as follows:

- **Machine Learning Algorithm**: Random Forests
- **Feature Selection**: Information Gain
- **Classifier Structure**: Cascade (Domain Knowledge)
- **Unbalanced Data**: Cost Sensitive

The predicted s/n ratio for this machine learning model is -119.1 db. Lastly, we perform a verification experiment to confirm that the optimal settings do produce machine learning model with a s/n ratio that is relatively close to the predicted s/n ratio and is larger than the experimental maximum s/n ratio of -119.284 db in Experiment 9. Using the false negative (FN) and false positive (FP) values from 5.7, we obtain a s/n ratio of -116.031 db. Since the s/n ratio of the predicted optimal is significantly larger then the experimental maximum (-119.284) db, we have confirmed that the predicted settings are actually optimal.

5.6 Results: Predicting Service Request Escalation in the Computer Networking Domain

In this section we examine the results for service request escalation problem in order to evaluate the effectiveness of the statistical Design of Experiments (DOE) approach to machine learning model selection. To this end, the results are organized into two parts. First, we review the results with respect to the problem‘s performance metric of interest, the total cost associated with a set of service requests. We then review the results with respect to a set of more general performance metrics that are independent to the service request escalation problem. Based on these two sets of results, we discuss the effectiveness of the DOE approach and draw conclusions.

Experiment	Total Cost ($M)	Accuracy	f-measure	g-mean
E^1	816.3	89.73	94.00	69.14
E^2	1109.3	85.40	91.50	76.67
E^3	814.7	89.70	94.00	71.60
E^4	1306.5	82.50	89.80	80.40
E^5	1530.2	82.50	89.80	80.40
E^6	1306.5	85.71	91.70	78.24
E^7	434.4	94.10	96.40	95.90
E^8	399.3	94.55	96.70	96.43
E^9	355.2	95.10	97.00	96.80
E^*	257.5	96.62	98.27	98.28

Table 5.9: Evaluation of the service request prediction models with respect to total cost, accuracy, f-measure, and g-mean

The results for each experiment with respect to our objective of minimizing total cost are shown in Table 5.9. First, we note that the DOE approach produces an combination of values for the external model parameters that is optimal with respect to the total escalation cost. We further note that the combination of parameters for the optimal model is not in the set of parameters tested in the nine experiments.

Second, we note that although the difference between the optimal model and best experimental model was relatively small ($\$97.5M$); the difference between the average ($\$639.5M$) and worst case experimental model ($\$1246M$) is significantly larger. This indicates that there is large amount of variability in the total cost based on the external parameters and learning algorithm used to create predictive model.

Third, we note that the experimental results show that the models that used the Random Forest learning algorithm (E^7, E^8, E^9) clearly out-performed the models using the naive bayes and support vector machines learning algorithms. The DOE approach allows a more comprehensive understanding of the experimental results where the user can quantitatively assess the influence of the different external parameters in the machine learning model. Using Table 5.8 we can draw the following conclusions about the variation in the performance of the models with respect to total cost:

1. The machine learning algorithm has the largest impact on the overall escalation cost.

2. Using a classifier cascade based on domain knowledge provides a slight benefit over a flat or boosted classification structure.

3. Cost sensitive learning did not make a significant difference. This is surprising considering that the costs are also part of the quality characteristic we are trying to minimize.

4. The use of domain knowledge models is beneficial for more complex machine learning tasks such as creating a multi-stage classifier. However, simpler tasks, such as feature selection, can successfully be automated using unsupervised machine learning without creating domain knowledge models.

Table 5.9 also shows the evaluation of the nine experiments and using accuracy, f-measure, and g-mean—three well-known evaluation metrics that are commonly used to evaluate the performance of machine learning models. Accuracy is the most commonly used single metric for evaluating machine learning models but is not necessarily an appropriate metric when the data-set is unbalanced because false positives and false negatives are equally weighted. We therefore also include the g-mean (geometric mean) and f-measure which are based on the precision and recall of the model. We note that the results for these metrics shown are consistent with total cost results discussed earlier. The optimal model has the best performance in the experimental space on all three evaluation measures. Most importantly, the optimal classification model outperforms the baseline experiments for both the f-measure and g-mean evaluation metrics.

The evaluation of the provided in Table 5.9 demonstrates that the DOE approach produces the optimal model with respect to the total cost performance metric for the service request escalation problem. As mentioned in Section 5.4, model selection is usually a compromise between efficiency and comprehensively exploring the experimental space. For the service request problem it was not feasible to run the 81 experiments necessary for a factorial design. Although, heuristics would have allowed us to reduce this search space, it would have been at the expense of being comprehensive. The DOE approach allowed us to balance being comprehensive with being efficient. As a result we were able to not only identify the optimal model for the service request escalation problem but also determine the theoretical performance bounds for a model in the designated parameter space.

5.7 Conclusions

In this chapter we have formalized the notion of an external parameter vector for the machine learning model that represents the engineering application specific decisions that the user needs to make in the machine learning process, such as feature selection and the creation of the training/test data-set. We then developed a simple process based on the Taguchi Design of Experiments (DOE) methodology to solve the model selection problem with respect to these external parameters including the learning algorithm.

We have demonstrated the application of the Taguchi based DOE approach to machine learning model selection in the context of a real-world problem in the computer networking domain which involved predicting service request escalation. For the service request escalation problem we found that being able to comprehensively explore the external parameter space resulted in a significant improvement in the classification function‘s performance with respect to the objective function of minimizing total cost as well as standard metrics such as accuracy and F-Measure.

For the problem of predicting service request escalation, we found that a statistical DOE based approach to optimizing the external parameters was a practical alternative in between an ad-hoc build-test-fix approach and using a full factorial experimental design to test all possible models. One of the key advantages to the statistical DOE method is it can simplified depending on the complexity and resource constraints of the problem under consideration. For simpler problems or when there are time constraints, we can assume no interaction between the factors and apply the approach described in Section 5.4. If additional time and resources are available, the user can relax this assumption and apply more sophisticated DOE techniques [Phadke, 1989] that test for interactions and then adjust the experimental design accordingly.

6 Developing High-Value Knowledge Engineering Software Products: Theory, Application, and Implementation

In this chapter we address the application of the representation-based framework, developed in Chapter 3, to the problem of developing knowledge engineering software products. Knowledge engineering processes are often largely manual and extremely inefficient in both cost and time. Therefore, software automation of these manual activities through the creation of highly user-centric Knowledge Engineering Software Products (KESPs) is critical to enabling the rapid and efficient extraction of high-quality knowledge.

The primary intent of this chapter is to provide a comprehensive theory, including its application and implementation, for developing high-value KESPs. The application of the IMRM to the development of high-value KESPs resulted in the Integrated Representation-Based Process Methodology (IRPM) which combines, in a rational and structured manner, methods and tools from the technical domains of Knowledge Engineering, Product Design, and Software Engineering. We demonstrate the feasibility of the IRPM by implementing it within the context a real knowledge engineering problem involving the extraction of problem-solution pairs from customer service requests in order to create "smart" products and services. The developed KESP, called the "Service Request Portal" (SRP), used problem-specific search and content filters which achieved a 30% productivity improvement over the previously manual work process.

The chapter is organized as follows. Section 6.2 develops three simple requirements for high-value Knowledge Engineering Software Products (KESPs) and motivates the need for methods and tools from multiple domains. Section 6.3 surveys related work in the three KESP subject matter domains: knowledge engineering, product design, and software engineering. Section 6.4 describes the application of the Integrated Meta-Representational Model (IMRM) to KESP development. Section 6.5 provides the process for applying the resulting representation-based model and demonstrates its implementation to a knowledge engineering problem at a large computer-network company. Section 6.7 provides the evaluation of the developed KESP with respect to the three requirements for high-value.

6.1 Introduction

This chapter addresses the design and development of high-value Knowledge Engineering Software Products (KESPs). The purpose of this section is to properly motivate this problem, outline the research issues involved, and describe our key contributions.

6.1.1 Background

Rapid and cost effective knowledge engineering requires the automation of largely manual activities through the creation of user-centric Knowledge Engineering Software Products (KESPs). Unlike commercially available data mining software, KESPs are custom software products that must be specific to the knowledge engineering process that needs to be automated. Therefore, the development of these KESPs involves three high-level tasks:

1. Modeling the activities in the currently manual knowledge engineering processes that need to be automated.
2. Designing high-quality, cost effective, products for automating the manual activities.
3. Developing software product implementations that are easy to use, reliable, and maintainable.

It is important to differentiate the "modern" KE systems addressed in this thesis from "classical" KE systems such as expert systems. Classical KE systems primarily address the problem of codifying human knowledge so that can be reused throughout the organization. We are interested in "modern" KE system that provide software automation to support existing work process and enable knowledge workers to efficiently work with massive amounts of data. Because modern KE systems involve integrating software systems into existing work processes, they share a number of similarities with Decision Support Systems (DSSs). However, DSSs generally operate with structured data, while modern KE systems support processes involving unstructured data. Table 6.1 highlights some of the key differences between classical KE systems, modern KE systems, and DSSs.

	Modern Knowledge Engineering System	Classical Knowledge Engineering system	Decision Support System
Knowledge source	Unstructured data	Domain experts	Structured data
Type of problems	Unstructured	Structured	Semi-structured
Users	Domain experts	Non-experts	Managers
Level of interactivity	High	Low	High
Computational requirements	Medium - high	Low - medium	Medium - high
Development challenges	Work-process integration (usability, adoption, etc.)	Knowledge capture and representation	Data integration

Table 6.1: Comparison of modern Knowledge Engineering systems, classical knowledge engineering systems, and decision support systems

Three important requirements for successful Knowledge Engineering (KE) system development are as follows: First, the KE system must be tightly integrated into the existing work processes in order to maximize the overall productivity of the end-users. Second, the KE system must be a high-quality product—reliable, easy to use, attractive—in order to be adopted by the end-users (knowledge workers). Third, the KE system must be high-impact and low-cost in order to a good investment for the organizations. Addressing of these three requirements requires a multi-disciplinary approach based on the following three domains:

- **Knowledge Engineering**: provides tools for modelling end-users work processes in order to tightly integrate the system into existing work processes.
- **Product Design and Development**: provides formal tools for explicitly identifying end-user needs for the system, exploring different function realizations, and managing the inevitable trade-off between quality and cost.
- **Software Engineering**: provides tools and techniques for efficiently developing robust and reliable software systems.

While there are a number good methodology for classical KE system development, such as CommonKADS [Schreiber, 1994] and MIKE [Angele et al., 1992], there is a lack of mature methodologies for developing modern Knowledge Engineering systems. Consequently, the selection and application of methods from these three domains is often ad-hoc, and, being ad-hoc, suffers from a number of issues including: insufficient capture the end-users' existing work-process, focus on the technical aspects of the system rather than users' needs, and lack of user involvement during system development. The combination of these factors, in particular the lack of attention to the user and organizational needs for the system, frequently results in the deployment of these system not yielding useful results.

6.1.2 Research Issues

Given that several diverse engineering disciplines are required to successfully address the KESP development tasks, described in the previous section, this problem involves research issues at three levels:

1. **Theory**: The formulation of a general framework, for organizing the activities—defining requirements, system design, software development, testing, etc.—and the engineering subject-matter domains—knowledge engineering, product design and

development, and software engineering—involved in the development of Knowledge Engineering Software Products.

2. **Application**: The specialization of the high-level framework to generate an integrated process methodology for developing Knowledge Engineering Software Products.

3. **Implementation**: The validation of the process methodology within the context of a real knowledge engineering problem in industry.

6.1.3 Contributions

The contributions of our research directly address the three levels of research issues—theory, application, and implementation—discussed in the previous section. The first contribution provides a general theory for selecting and integrating methods and tools from multiple domains. The second and third contributions are specific to application and implementation of this theory for the purposes of developing high-value Knowledge Engineering Software Products.

1. **Theory**: We have applied the Integrated Meta-Representational Model to first identify, and then select methods and tools from three engineering subject-matter domains—knowledge engineering, product design, and software engineering domains—for the purposes of addressing the development of knowledge engineering software products. (See Section 4.4.)

2. **Application**: We developed a process methodology, called the Integrated Representation-Based Process Methodology (IRPM), that provides the theoretical framework for using the selected methods for the purpose of developing Knowledge Engineering Software Products (Section 6.5). The knowledge engineering domain provides tools—such as the CommonKADS Agent/Task model—for modeling current work

processes that the KESP will automate. The product design domain provides formal tools—such as the House of Quality, Function Structure, Morphological Matrix, and Utility Function—for explicitly defining the user needs for the KESP and exploring different design concepts in order to ensure the KESP is high-quality and low-cost. The software engineering domain provides tools—such as the UML Use Case, Component, and Class diagrams—in order to ensure that a reliable and user-centric KESP is delivered rapidly and at low cost. We have also shown how the IRPM can be simplified for situations where there are time constraints or the knowledge engineering process being automated is not overly complex (Section 6.7.4).

3. **Implementation**: We have demonstrated the feasibility of the IRPM within the context of a knowledge engineering problem involving the extraction of problem-solution pairs from customer service requests in order to create "smart" products and services. The developed KESP, called the "Service Request Portal" (SRP), improved productivity by over 30% compared to the previously used tools, was well received by the users, and was developed within the organizations budget and schedule constraints. (Section 6.7).

6.2 Problem Formulation

In this section we establish the need for a multi-disciplinary approach to the development of Knowledge Engineering Software Products (KESPs), and outline the tasks involved in creating such an approach. We start with a brief discussion of the requirements for high-value KESPs. Next, we use a real-world knowledge engineering problem in the computer-networking domain to concretely illustrate the issues and challenges that come up when attempting to satisfy these requirements in practice. Further examination of these issues motivates the need for a multi-disciplinary approach involving methods and techniques from the domains of knowledge engineering, product design, and software engineering. We conclude by describing the tasks involved in creating and applying a theoretical framework to integrate the methods from these three domains.

6.2.1 Three Requirements for High-Value Knowledge Engineering Software Products

This chapter addresses the development of high-value Knowledge Engineering Software Products (KESPs) that automate currently manual knowledge engineering processes in order to produce higher-quality solutions faster and at lower cost. To this end, we have developed three simple requirements that can be used to assess whether a KESP delivers high-value.

First and foremost, the KESP must be high-quality with respect to the user needs. Here, high-quality has two aspects. The first part (requirement *R1a*) addresses the integration of the KESP with the existing knowledge engineering work processes. The second part (requirement *R1b*) addresses the user experience with the KESP.

Requirement 1a (R1a): The KESP must be seamlessly integrated with the current, largely manual, end-user work process.

Requirement 1b (R1b): The KESP must be reliable, high-quality, and easy to use.

The second set of requirements relate to the direct impact of using the KESP on the productivity of the end-users (requirement *R2a*) and to the indirect impact of using the KESP on the organization (requirement *R2b*).

Requirement 2a (R2a): The KESP must improve the productivity of the end-users, e.g. enable higher-quality results to be produced significantly faster.

Requirement 2b (R2b): The use of the KESP must help the organization develop higher-quality and/or lower cost products and services.

Finally, it is also necessary to balance quality with development costs (time and resources) to ensure the KESP is high-value.

Requirement 3 (R3): The KESP must be developed rapidly and at low cost.

In order to concretely illustrate the three requirements for high-value consider the following knowledge engineering problem at a large company that develops and supports enterprise computer networking products. Every day the company's service centers receive thousands of customer service requests (commonly referred to as "trouble tickets") on a wide variety of product problems. Each service request is assigned to a Technical Support Engineer who works with the customer to resolve the problem. After the problem is resolved the associated service request document containing the correspondence (emails, phone calls, etc.) between the customer and the Technical Support Engineer is archived in a relational database. This database contains millions of service requests that can be used to improve product support for existing products as well as to design new products.

The company has a team of Network Knowledge Engineers (NKEs) that search through the resolved service requests in order to extract problem-solution pairs for frequent customer problems. These problem-solution pairs are then used by the company to improve product support and develop new product. The current, largely manual, process for extracting problem-solution pairs consists of two high-level activities: searching for relevant service requests to a particular problem, and reading the relevant service requests in order to extract problem-solution pairs. The NKEs currently use a search engine to perform keyword searches in order to locate relevant service requests, and a web-based viewer to read the service requests.

The NKEs' manual work process is very inefficient because of the following problems:

1. **Precision problem**: Keyword searches typically return in an large number of search results that are time consuming to evaluate for relevance. Furthermore,

keyword searches can not be narrowed down using service request attachments such as device configuration files.

2. **Summary problem**: The service request documents do not include a clear description of the solution that resolved the customer's problem.

3. **Repetition**: The service request documents typically contain a large amount of repetition in the form of repeated email threads and redundant case notes.

The "precision" problem impacts the information retrieval aspects of the NKE work process and requires the NKEs to manually evaluate a large number of irrelevant search results. The "summary" and "repetition" problems impact the extraction of problem-solution pairs from relevant service requests. In particular, the long length of each service request (typically 30-50 pages of free-form text) in combination with the presence of irrelevant email threads, poorly formatted text, and duplicate content make reading each service request difficult and time consuming.

From the perspective of the NKEs, the KESP problem statement is as follows: develop a software product to automate the tedious and manual aspects of extracting problem-solution pairs. In particular, the NKEs would like the software product to make it easier to find relevant service requests for a particular product problem (the "precision" problem) and extract the problem-solution pairs from a given service request (the "summary", and "repetition" problems).

From the perspective of the networking company, the KESP problem statement is as follows: develop a software product to improve the productivity of the NKEs and decrease the cost of developing problem-solution pairs.

6.2.2 The Multi-Disciplinary Approach

Developing a high-value Knowledge Engineering Software Product (KESP) to address the NKE's problem and the networking company's problem requires methods and techniques from three related but distinct subject matter domains. First, in order to create an integrated product (requirement *R1a*) that significantly improves the productivity of the NKEs (requirement *R2a*) we need formal methods from the knowledge engineering domain to model the work-process for developing problem-solution pairs. Second, developing a high-quality product (requirements *R1b* and *R2a*) requires product design methods to tightly couple the NKEs in the KESP design and development process. Third, in order to rapidly build the KESP at low cost (requirement *R3*) we need software engineering methods to manage the software design and development processes.

The selection and application of methods and techniques from the knowledge engineering, product design, and software engineering domains involves the following tasks:

1. Create a model for the activities involved in KESP development. (Section 6.4)
2. Apply the model to select the necessary knowledge engineering, product design, and software engineering methods for KESP development. (Section 6.4)
3. Integrate the selected methods into a process methodology for KESP development. (Section 6.5)
4. Demonstrate the feasibility of the process methodology within the context of a real knowledge engineering problem and assess the results (Section 6.5 and Section 6.7).

6.3 Literature Survey

In this section we survey related work in the three subject matter domains involved in the development of Knowledge Engineering Software Products (KESPs): Knowledge

Engineering, Product Design, and Software Engineering. Since many readers may not be familiar with all three of the domains, we start with a brief background to some of the key methods for each domain. We then expand on the strengths and weakness of each domain in order to further motivate the need for all three domains when developing high-value KESPs.

6.3.1 Knowledge Engineering

The Knowledge Engineering domain encompasses a large body of work spanning many different areas including: artificial intelligence [Winston, 1992], knowledge representation [Brachman and Levesque, 2004], expert systems [Schreiber, 1994], data mining [Witten and Frank, 2005], and information retrieval [B. et al., 2009]. Our work draws on three key knowledge engineering areas: expert systems, information retrieval, and data mining. We use methodologies from expert systems to model the currently manual work processes for knowledge extraction. Since Knowledge Engineering Software Products generally contain either information retrieval or data mining components, or a combination of the two, we also draw on standard methods and techniques from information retrieval and data mining to implement the KESP. The specific KESP described in this work (Section 6.5) contained information retrieval functionality that was layered on top of search engine infrastructure that already existed at the organization.

CommonKADS [Schreiber, 1994] is a well-known methodology for creating expert systems that consists of three perspectives (sets of models) that guide the design and development of expert systems. The first perspective addresses the organizational environment in which the expert system will operate and is used to understand the objectives for the expert system and how it will fit into existing work processes. The second perspective addresses the knowledge components of the system and provides models for identifying and structuring the domain expert knowledge necessary for solving a particular task. The third perspective addresses the design of the system architecture and computational mechanisms for expert system.

CommonKADS, and other related modeling methodologies, provide methods for identifying organizational opportunities for the KESP development process and structuring end-user work processes (requirements *R1a*, *R2a*, *R2b*). However, these methodologies generally have limited support for implementing these models into software products. In particular, the models are generally not integrated with standard software engineering tools, such as the Unified Modeling Language, which makes it difficult to develop KESP software implementations.

There are several expert system methodologies that specifically address the implementation aspects of the development process. The Model Based Incremental Knowledge Engineering (MIKE) methodology [Angele et al., 1992] adds a formal specification language, called KARL, to the CommonKADS models which enables rapid prototyping and incremental development. However, this specification language is specific to expert systems and not suitable for KESP development.

6.3.2 Product Design

In most organizations, once knowledge engineering issues are addressed, development typically proceeds directly to addressing software engineering issues. In our view this fails to leverage valuable tools in the product design domain that enable different design concepts to be generated and assessed with respect to development trade-offs. In this section we provide a brief overview of the Product Design methods for conceptual design and discuss the strengths and weaknesses of these methodologies with respect to the three value requirements developed in Section 6.2.

Product Design methods for conceptual design fall into three general areas: specifying the design space, exploring the design space (generating alternative design concepts), and selecting design concepts. Two well-known methods for specifying the design space are the House of Quality [Hauser and Clausing, 1988] and Function Structure [Pahl and Beitz, 1996] methods. The House of Quality relates user (customer) needs to measurable engineering metrics that can be used to design the product. The Function Structure method establishes the functional (input-output) relationships between the requirements for the product. Once the requirements for the product are specified, the Morphological Matrix [Pahl and Beitz, 1996] provides a systematic approach for exploring the space of possible solution-principles (realizations) and combining solution principles in order to generate alternative design concepts. The Utility Function method [Pahl and Beitz, 1996] then provides a quantitative approach to comparing the utility of alternative design concepts with respect to selection criteria (objectives) in order to select the concept that maximizes the cumulative utility across all the selection criteria.

The core product design methods described above have been integrated into a number of well-known methodologies for designing new products. The Engineering Design methodology [Pahl and Beitz, 1996], illustrated in [Hubka et al., 1988], provides a functional approach—based on Function Structures, Morphological Matrices, and Utility Functions—for developing a design concept for the product. The Product Design and Development methodology [Otto and Wood, 2000] builds on the functional approach in the Engineering Design methodology by adding the House of Quality and other techniques for capturing user needs for the product. The Total Design methodology [Pugh, 1991] uses a process called controlled convergence, based on the utility function method, to iteratively generate and evaluate alternative designs and converge on a design concept for the product.

The product design domain provides structured methods and tools for capturing user needs and using these needs to guide the design of the product. In particular, the product design methods allows for the trade-offs between quality and cost to assessed in an analytical manner. However, most of these methods were originally created for designing physical products and do not directly address important of software products such as data structures. Consequently, these product design methods need to be combined with software engineering methods and techniques in order to develop software products.

6.3.3 Software Engineering

A wide range of software engineering methodologies have been proposed over the last 30 years. The Waterfall model [Schach, 2008] uses a sequential development process composed of five stages: requirements, design, implementation, verification, and maintenance. The Spiral model [Schach, 2008] expands on the Waterfall model and adds support for risk analysis and prototyping. Iterative and Incremental Development [McConnell, 1996] uses increments, rapid build and test cycles, with user feedback to iteratively develop software. Agile Processes [Schach, 2008]—such as Extreme Programming (XP), Crystal, and Scrum—expand on Iterative and Incremental Development with timeboxing and user participation during the design process. All of these methodologies are supported by the Unified Modeling Language (UML) [Schach, 2008] which provides a standard specification language for software systems.

These software engineering methodologies provide methods for developing high-quality software with respect to the user experience (robust, reliable, easy to use). However, these methodologies generally follow a direct path between the requirements for the system and implementation of the system and do not include a conceptual design stage where multiple alternative approaches for the implementing the specifications are explored and evaluated. Although this direct approach is sufficient for implementing straightforward technical systems, it generally is not sufficient for KESP development which involve complex value trade-offs between the needs of the end-users and the needs of the organization.

It is important to note, there are software engineering methodologies that address these gaps by providing tools for capturing existing work processes (requirements *R1a*, *R2a*, *R2b*) and balancing quality with costs (requirement *R3*). Jackson System Design [Jackson, 1983] and Problem Frames [Jackson, 2001] provide methods for analyzing and structuring software development problems. Value Based Software Engineering (VBSE) [Boehm, 2003] addresses some aspects of the conceptual design stage by integrating value considerations into existing software engineering methodologies through the use of practices such as Business Case Analysis, and Concurrent Engineering. While these methodologies are useful, they do not provide comprehensive (end-to-end) approach to addressing the value requirements. For example, both Jackson System Development and Problem Frames contain tools for modeling the relationships between the software system and the end-users, but lack tools for modeling user work-processes. Similarly, VBSE introduces useful practices for addressing value considerations, but lacks methods for generating multiple design concepts—a critical step in developing high-value KESPs.

6.4 Theory: Representation-Based Model for the Development of Knowledge Engineering Software Products

The application of the Integrated Meta-Representational Model (IMRM) facilitated the integration of methods and tools from knowledge engineering, product design, and software engineering domains into the Integrated Representation-Based Process Methodology (IRPM) for the development of high-value Knowledge Engineering Software Products (KESPs). This section describes the results for each of the three steps in applying the IMRM.

Table 6.2 shows the layers, representational subject matters, domains, and methods for KESP development. The *External* layer uses the Organization and Agent/Task CommonKADS models [Schreiber, 1994] from the knowledge engineering domain to model the currently manual work process that needs to be automated. The *Outside-In* layer uses the House of Quality method [Hauser and Clausing, 1988] and UML Use

Layer	Subject Matter	Domains and Representational Methods/Tools		
		Knowledge Engineering	Product Design	Software Engineering
External	Organizational context and currently manual Knowledge Engineering work process	CommonKADS Organization, Agent, and Task Models		
Outside-In	User needs for software automation		House of Quality	Use Case Diagrams
Internal	Conceptual design (functional specifications, solution-principles, design concept for the product)		Function Structure, Morphological Matrix, Utility Function	
Inside-Out	Software design (architecture, data structures, algorithms, control logic)			UML Component and Class Diagrams
Outer	Software development (planning, implementation, and testing)			Iterative and Incremental Development

Table 6.2: Layer-subject matter-representational tools/methods matrix for knowledge engineering software products

Case diagrams [Schach, 2008] to capture the end-user needs and desired user experience for the KESP. The *Internal* layer uses the Function Structure [Pahl and Beitz, 1996], Morphological Matrix [Pahl and Beitz, 1996], and Utility Function [Pahl and Beitz, 1996] methods from the product design domain in order to explore different function realizations for the software automation and manage the inevitable trade-offs between quality and cost. The *Inside-Out* layer uses the Unified Modeling Language Component and Class diagrams [Schach, 2008] from the software engineering domain to create the software design for the KESP. The *Outer* layer uses the Iterative and Incremental development methodology [McConnell, 1996] from the software engineering domain to develop the KESP software implementation.

Figure 6.1 shows representation-based model that resulted from integrating the methods and tools from Table 6.2. Each information flow is depicted as directional arrow that shows the relationship between the inputs and outputs of two methods (with the exception of information flow (7) which involves the output of two different methods). This input-output integration of the methods and tools allows for the three KESP value requirements (Section 6.2) to be incrementally addressed as we progress from the *External* to the *Outer* layer. Each layer adds value by either aligning the results from the previous layer with one or more of the value requirements and/or implementing the value created a previous layer.

The representational-based model is implemented by sequentially stepping through each layer shown in Figure 6.1 starting with the initial state or high-level user need as follows:

1. ***External* layer**: take the high-level user need (see (1) in Figure 6.1) from the initial state as input and create the CommonKADS Organization and Agent/Task models of the current work processes. These models address value requirements *R1a*, *R2a*, *R2b* by aligning the KESP with the high-level user needs and work process.

2. ***Outside-In* layer**: use the work process model (2) to create a House Quality and set of Use Case diagrams for the KESP. The *Outside-In* layer captures what the end-users want from the product and ensures that the KESP is high quality with respect to the user needs (value requirements *R1a* and *R1b*).

3. ***Internal* layer**: use the user requirements (3)(5) and Use Case diagrams (4) to create the Function Structure, Morphological Matrix, and Utility Function for the KESP. The *Internal* layer methods address value requirements *R1a*, *R1b*, *R2a*, *R2b*, and *R3* by ensuring that the KESP is functionally complete and provides a good balance of quality and cost with respect to the user needs.

4. ***Inside-Out* layer**: create a UML Component diagram and corresponding set of UML Class diagrams to transform the design concept (6)(7) into a software

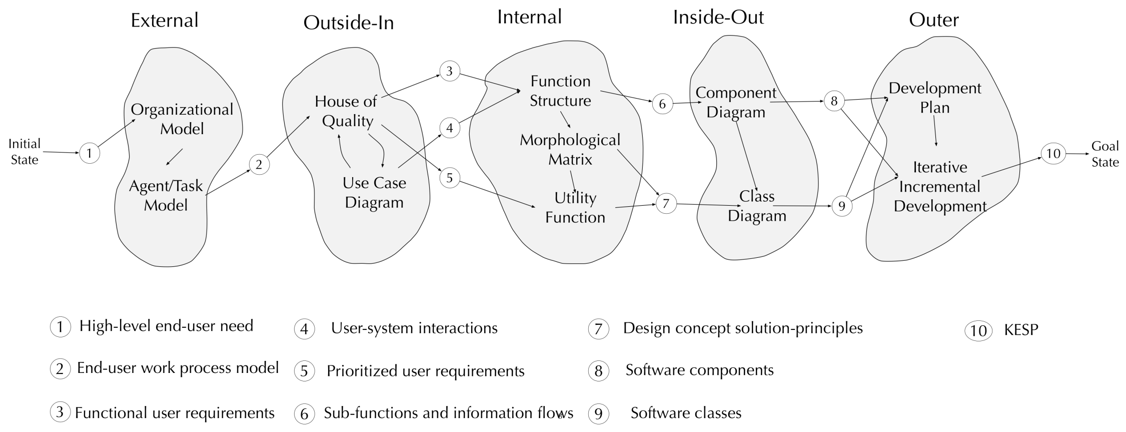


Figure 6.1: Representation-based model for knowledge engineering software product development

design for the KESP. The *Inside-Out* layer indirectly addresses requirements *R1a*, *R1b*, *R2a*, *R2b*, and *R3* by ensuring the product design results from the *Internal* layer are accurately translated into Software Engineering constructs that can be developed at the *Outer* layer.

5. ***Outer* layer**: implement the software architecture (8) and classes (9) to produce the KESP that satisfies the goal state (10). The *Outer* layer ensures that the KESP addresses value requirements *R1a*, *R1b*, *R2a*, and *R2b*.

The comprehensive (step-by-step) implementation of the IRPM is provided in the following section (Section 6.5).

6.5 Application: Process Methodology for Knowledge Engineering Software Product Development

In this section we provide the step-by-step process for implementing the five layers of the representation-based model shown in 6.1. The methods and techniques at each layer are illustrated using the quality monitoring and assessment problem described in

Section 6.2 involving the ASA 5505 computer network security product.

6.5.1 External Layer: Modeling the Organization Context

The development of a Knowledge Engineering Software Product (KESP) starts with a currently manual knowledge engineering process that needs to be automated. For example, the development of the Service Request Portal (SRP) started with the need to automate the process of extracting problem-solution pairs. The purpose of the *External* layer is to model the currently manual work process in order to explicitly identify the activities that will benefit from automation.

The *External* layer makes use of three models from CommonKADS methodology [Schreiber, 1994] in order to address the integration of the KESP with the currently manual work process (requirement *R1a*) and its use by the knowledge engineers (requirements *R2a*, *R2b*). The Organization model is used to align the KESP the organizational problem that needs to be solved in order to satisfy requirement *R2b*. The Agent and Task models are used to capture the end-users' work process in order to ensure that the KESP is integrated into the end-users' work process (requirement *R1a*) and improves productivity (requirement *R2a*).

The *External* layer process for applying the Organization and Agent/Task models is as follows:

1. Interview stakeholders (typically this includes end-users and their managers) about the high-level user and organization need(s) for the KESP. Important information to collect includes current work processes, the people and resources involved in these work processes, and success metrics for the KESP. These inputs to the *External* layer are indicated by connection (1) in Figure 6.1.

2. Organize the collected information using a **CommonKADS Organization model**

[Schreiber, 1994] and determine the focus area for the KESP. Figure 6.2 shows the Organization model for the SRP. The process for creating the Organization model is as follows: a) Extract the key organizational components—context, people, processes, resources—from the information collected in Step 1; b) Identify problems in the current processes; c) Select one high-value problem to be the focus area for the KESP.

3. Interview the end-users about their work process and record the results as a step-by-step process. Techniques for work process interviewing are discussed in [Schreiber, 1994].

4. Construct an **Agent/Task model** [Schreiber, 1994] for the current work process. The Agent/Task model for the Service Request Portal is shown in Figure 6.3. The process for creating the Agent/Task model is as follows: a) Create an agent for each person or resource involved in the current work process; b) Create a task for each step in the work process captured in Step 3; c) Relate agents to the tasks that they perform. Label tasks performed by a single actor as $<<$ *includes* $>>$ and tasks performed by multiple actors as $<<$ *uses* $>>$.

5. Use the Agent/Task model at the *Outside-In* layer to identify problems in the currently manual work process. This output from the *External* layer is indicated by connection (2) in the IRPM process shown in Figure 6.1.

The *External* layer for the Service Request Portal development process consisted of the Organization model shown in Figure 6.2 and Agent/Task model shown in Figure 6.3. The Organization model ensured that the SRP significantly accelerated the process of extracting problem-solution pairs (requirement *R2b*) by identifying the following important problem in the networking company: "locating relevant service requests and extracting problem-solution pairs". The Organization model also identified the search engine and service request viewer that were used in the SRP design concept in order

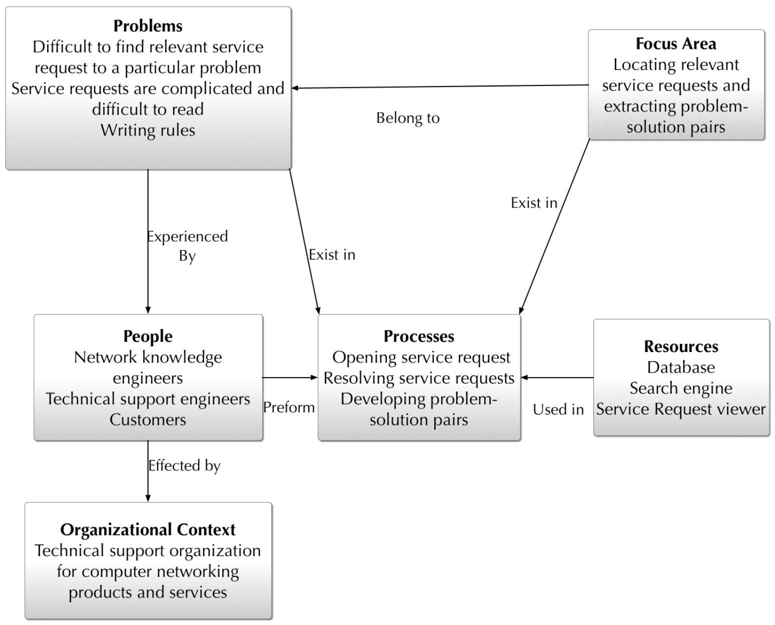

Problems
Difficult to find relevant service request to a particular problem
Service requests are complicated and difficult to read
Writing rules
Focus Area
Locating relevant service requests and extracting problem-solution pairs
Belong to
Exist in
Experienced By
Exist in
People
Network knowledge engineers
Technical support engineers
Customers
Preform
Processes
Opening service request
Resolving service requests
Developing problem-solution pairs
Used in
Resources
Database
Search engine
Service Request viewer
Effected by
Organizational Context
Technical support organization for computer networking products and services

Figure 6.2: CommonKADS Organization Model for the computer networking company where the Service Request Portal was developed

to minimize cost and development time. The Agent/Task model provided an easy to understand representation of the currently manual NKE work process which in-turn guided the process of eliciting the user needs at the *Outside-In* layer. This coupling of the existing work process and user needs ensured that the SRP was correctly integrated into the NKEs' work process (requirement *R1a*) and made significant productivity improvements (requirement *R2a*).

Figure 6.2 shows the relationship between the six key components—Organizational Context, People, Processes, Resources, Problems, and Focus Area—for the computer networking company where the SRP was developed. This model was used to determine that "locating relevant service requests and extracting problem-solution pairs" would be the focus area for the SRP. It also identified the existing resources (the search engine and service request viewer) that were used in the NKE work process.

Figure 6.3 shows the Agent/Task model corresponding to the NKE work process for "locating relevant service requests and extracting problem-solution pairs". This

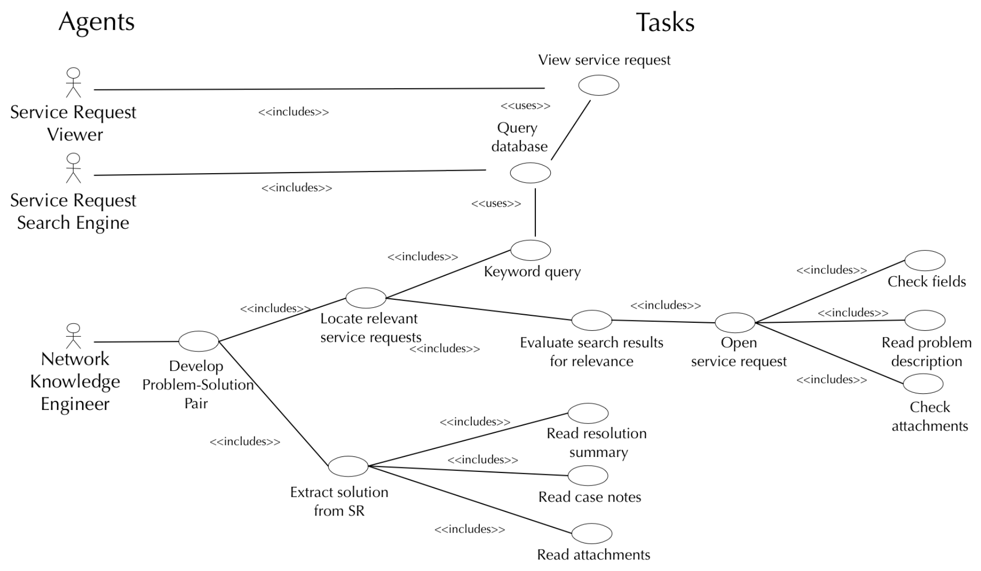


Figure 6.3: CommonKADS Agent/Task model of the Network Knowledge Engineers' work process for extracting problem-solution pairs

Agent/Task model facilitated the identification of the user needs for the Service Request Portal at the *Outside-In* layer and enabled the SRP development team to clearly visualize the relationships between the tasks in the NKE work process.

6.5.2 Outside-In Layer: User Needs and System Requirements

Once we have modeled the knowledge engineering work processes, we next need to understand the needs of the knowledge engineers who will be using the Knowledge Engineering Software Product (KESP). The purpose of the *Outside-In* layer is to identify user needs and correlate them to measurable development goals for the KESP. These user needs and goals are used at the *Internal* layer to help guide the conceptual design process and at the *Outer* layer to ensure that the KESP meets end-users' needs.

The *Outside-In* layer ensures a high-quality KESP with respect to the user needs (requirements *R1a* and *R1b*). To this end, the *Outside-In* layer combines techniques from the product design and software engineering domains. First, the House of Quality method [Hauser and Clausing, 1988], from the product design domain, is used to correlate the user needs with technical metrics (benchmarks) that can be used to develop the KESP. Next, the UML Use Case diagram [Schach, 2008], from software engineering domain, is used to specify the work flow for how the end-users will interact with the KESP.

The *Outside-In* layer process for applying the House of Quality and UML Use Case diagram methods is as follows:

1. Review the Agent/Task model from the *External* layer with the end-users and identify tasks that are currently difficult, problematic, and time-consuming. This input to the *Outside-In* layer is indicated by connection (2) in Figure 6.1.

2. Correlate the user needs and technical metrics for the KESP using the **House of Quality** method [Hauser and Clausing, 1988]. Figure 6.4 shows the House of Quality for the Service Request Portal. The process for creating the House of Quality is as follows: a) Work with end-users to define the functional requirements for the KESP; b) Interview end-users to identify the usability requirements, e.g. easy to use, for the KESP; c) Work with users to assign a relative priority or importance to each functional and usability requirement using a convenient scale e.g. 1-10; d) Derive a set of technical metrics for quantitatively measuring the satisfaction of the user needs based on the tasks in the current work process; e) Characterize the relationships between the user needs and technical metrics using a convenient scale (e.g. strong relationships, medium relationship, weak relationships, or no correlation); f) Characterize the correlation between the technical metrics using a convenient scale (e.g. positive, negative, or no correlation).

3. Translate each secondary functional user need—and all primary functional needs which do not decompose into secondary needs—in the House of Quality into a goal that the users are trying to accomplish using the KESP. Review the goals

with the end-users to ensure that they correctly reflect the intent of the original user need.

4. For each user goal create a **UML Use Case diagram** [Schach, 2008] to capture the desired the user-product interactions related to achieving the goal. Figure 6.5 shows one of the Use Case diagrams for the Service Request Portal. The process for creating the Use Case diagram for a single user goal is as follows: a) Represent each agent in the Agent/Task model involved in achieving the goal as an actor; b) Work with end-users to add the desired set of interactions for accomplishing the goal; c) Label each set of related interactions as either a dependency (indicated by << *includes* >> in UML notation) where one interaction includes the other or a variation (<< *extends* >>) where one interaction is a special case of the other.

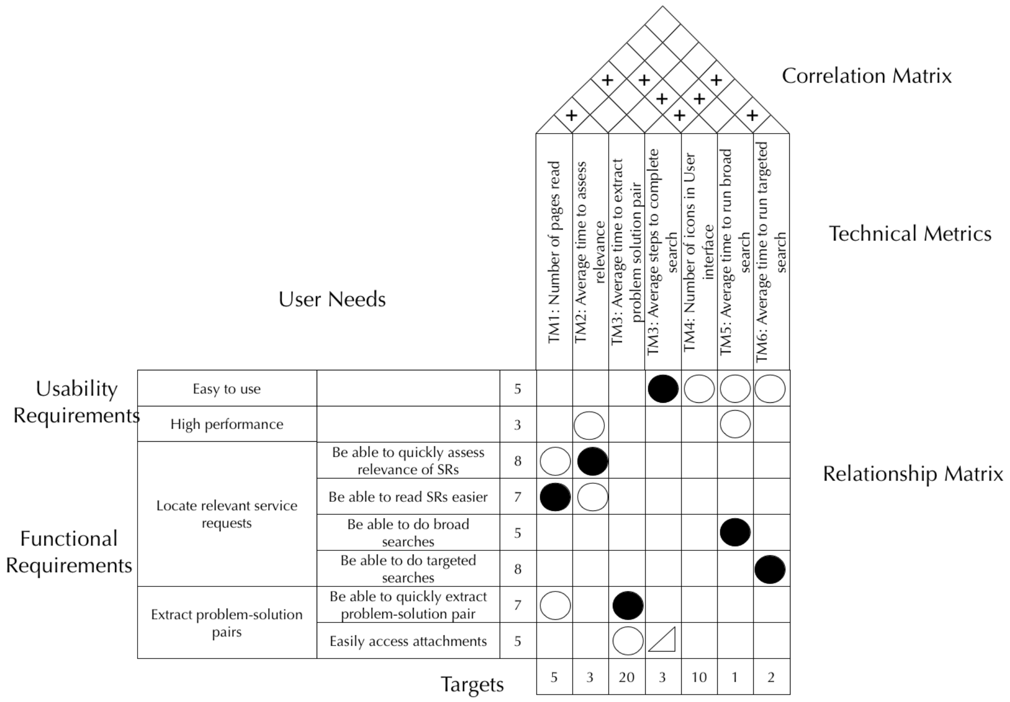
5. Review the Use Case diagrams with the end-users and, if necessary, make adjustments to the House of Quality.

6. Use the functional requirements from the House of Quality to specify the primary (overall) function of the KESP at the *Internal* layer. Use the interactions from the Use Case diagrams to define the inputs and outputs to the primary function. These outputs are indicated by connections (3) and (4) in the IRPM process (Figure 6.1). Use the functional and usability requirements to specify the evaluation criteria for the Utility Function at the *Internal* layer. These outputs from the *Outside-In* layer are indicated by connection (5) in Figure 6.1.

The *Outside-In* layer for the Service Request Portal consisted of the House of Quality diagram shown in Figure 6.4 and six Use Case diagram, one of which is shown in Figure 6.5. The House of Quality identified the NKEs' needs and was critical in ensuring the SRP was high-quality respect to the user needs (requirement *R1b*). At the *Internal* layer the House of Quality provided well-defined user requirements to guide the conceptual design process. The House of Quality also provided the technical metrics that served as evaluation criteria for guiding the implementation of the KESP at the *Outer* layer. The Use Case diagrams allowed the NKEs to work with the development team in order to ensure that the SRP was integrated into the existing work process (requirement *R1a*) and easy to use (requirement *R1b*).

Figure 6.4 shows the functional requirements, usability requirements and corresponding technical metrics for the Service Request Portal. The functional requirements were derived from the following two problems in the current work process. First, the relevance of a service request depended information—e.g. attachments and tags—that was not included in the existing search engine's keyword search functionality. Consequently, the task "keyword query" resulted in a large number of irrelevant results that needed to be manually evaluated for relevance. Second, each service request contained 30-50 pages of complex unstructured highly technical documentation. As a result, the NKEs spent a significant amount of time and effort reading through irrelevant email threads, poorly formatted text, and duplicated content when they performed the tasks: "read SR resolution summary", "read SR case notes", and "read SR attachments". These two problems were translated into the two primary functional requirements: "locate relevant service requests" and "extract problem-solution pairs". Each primary functional requirement was decomposed into a set secondary functional requirements based on the individual tasks in the NKE work process. For example, in order to locate relevant service requests the NKEs manually examined the fields (technology, sub-technology) of each service request as part of the task "check SR fields". The NKEs wanted to be able to specify these field values as part of the search criteria so that they did not have to manually filter search results. This user need was captured in the secondary functional requirements "be able to do targeted searches".

The two usability requirements—"easy to use" and "high-performance"—came from interviews with the NKEs. The technical metrics were based on the tasks in currently manual NKE work process and the functional requirement. For example, the "number



			TM1: Number of pages read	TM2: Average time to assess relevance	TM3: Average time to extract problem solution pair	TM3: Average steps to complete search	TM4: Number of icons in User interface	TM5: Average time to run broad search	TM6: Average time to run targeted search
Easy to use		5				●	○	○	○
High performance		3		○				○	
Locate relevant service requests	Be able to quickly assess relevance of SRs	8	○	●					
	Be able to read SRs easier	7	●	○					
	Be able to do broad searches	5						●	
	Be able to do targeted searches	8							●
Extract problem-solution pairs	Be able to quickly extract problem-solution pair	7	○		●				
	Easily access attachments	5			○	△			
Targets			5	3	20	3	10	1	2

Figure 6.4: House of Quality of the user needs and corresponding technical metrics for the Service Request Portal

of pages read to assess relevance" and "average time to assess relevance" were used to measure the functional requirement "be able to read service requests easier".

The correlation of the functional requirements, usability requirements, and technical metrics was based on discussions with the NKEs. For example, the functional requirement "be able to read service requests easier" was strongly related to the technical metric "number of pages read to assess relevance" because the NKEs found that shorter service requests were easier to read through. Likewise, the technical metric "number of pages read to assess relevance" was positively correlated to the technical metric "average time to assess relevance" because decreasing the number of pages that the NKEs had to read also decreased the average time to assess the relevance of a service request.

Figure 6.5 shows the use case corresponding to the functional requirement "be able to quickly assess relevance of service requests". This use case specified the desired interaction with the SRP when assessing the relevance of a service request. The direct result of this use case was the right pane of the user interface shown in Figure 6.11 which allowed the NKEs to quickly switch between the "summary", "tidy", and "original" views of the currently opened service request.

6.5.3 Internal Layer: Conceptual Design

The *External* and *Outside-In* layers provide the user and organizational needs that the Knowledge Engineering Software Product (KESP) will need to address. The *Internal* layer provides the design concept that specifies how the KESP will satisfy those needs. The resulting design concept is then translated into the software design for the KESP at the *Inside-Out* layer of representation.

The *Internal* layer makes use of three well-known product design techniques to address the three requirements for high-value (*R1*, *R2*, *R3*). First, the Function Structure method [Pahl and Beitz, 1996] is used to create a solution-neutral functional specifi-

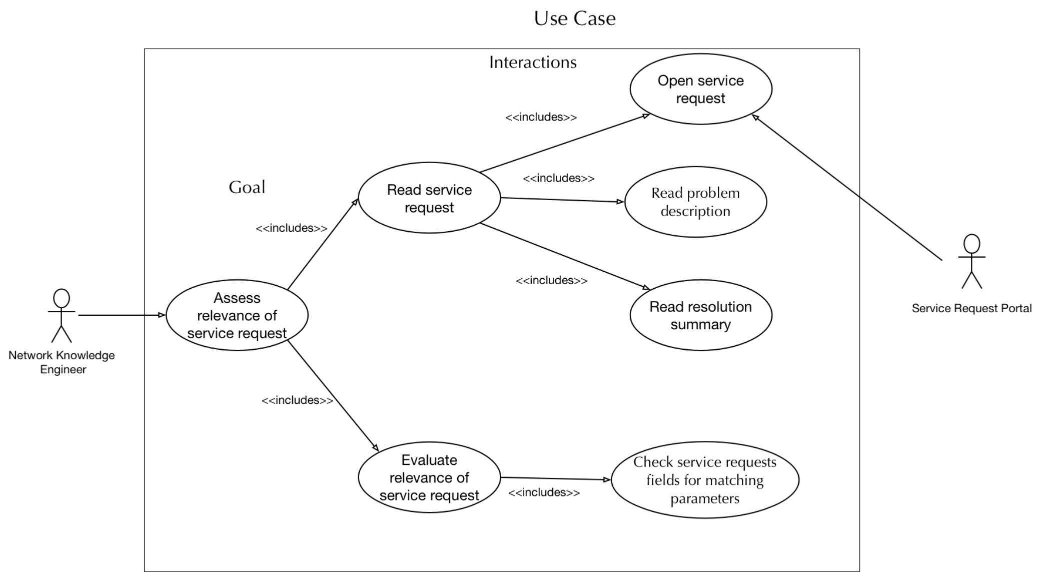

Use Case
Interactions
Goal
Open service request
Read service request
Read problem description
Read resolution summary
Assess relevance of service request
Evaluate relevance of service request
Check service requests fields for matching parameters
<<includes>>
Network Knowledge Engineer
Service Request Portal

Figure 6.5: Use Case diagram for the goal "assess relevance of a service request"

cation of the KESP. Next, the Morphological Matrix method [Pahl and Beitz, 1996] provides a systematic approach to exploring the design space and generating alternative KESP design concepts. Lastly, the Utility Function method [Pahl and Beitz, 1996] is used to assess the design concepts so that a single high-value concept can be selected for further development.

The *Internal* layer process for applying the Function Structure, Morphological Matrix, and Utility Function methods within the context of KESP development is as follows:

1. Determine the main function of the KESP from the functional requirements in the House of Quality. Examine the user-product interactions in the Use Case diagrams and determine the inputs and outputs to this main function. These two inputs to the *Internal* layer are indicated by connections (3) and (4) in Figure 6.1.

2. Create a Function Structure [Pahl and Beitz, 1996] for the KESP by hierarchically decomposing the primary function into lower complexity (simpler) sub-functions. Figure 6.6 shows the Function Structure for the Service Request Portal. The

process for creating the Function Structure is as follows: a) Decompose the main function (from Step 1) into a set of primary sub-functions corresponding to the primary functional requirements in the House of Quality; b) Decompose each of the primary sub-functions into a set of secondary sub-functions roughly corresponding to the secondary functional requirements in the House of Quality; c) Continue the decomposition process until each sub-function is atomic and only operates on a single input/output. The Agent/Task model of the user work process can be helpful in defining the lower level sub-functions past the level of detail in the House of Quality; d) Review the finished Function Structure with the end-users and verify that it accurately captures the requirements specified in the House of Quality and Use Case diagrams.

3. Create a Morphological Matrix [Pahl and Beitz, 1996] using the Function Structure from Step 2 and generate several alternative design concepts for the KESP. Figure 6.7 shows the Morphological Matrix that was used to generate the design concept for the Service Request Portal. The process for creating the Morphological Matrix is as follows: a) Identify several (2-5) solution-principles or function realizations for each terminal (bottom-level) sub-function in the Function Structure; b) Arrange the sub-functions and solution principles as a matrix, with sub-functions in the matrix rows and solution-principles in the matrix columns; c) Combine suitable combinations of solution principles to generate several (2-3) alternative design concepts for the KESP. Strategies for identifying solution-principles and generating design concepts are discussed in [Pahl and Beitz, 1996], [Ulrich and Eppinger, 1995], and [Otto and Wood, 2000].

4. Construct a Utility Function [Pahl and Beitz, 1996] to assess the design concepts for the KESP and select a single high-value concept for further development. Figure 6.8 shows the Utility Function that was used to select the design concept for the Service Request Portal. The process for applying the Utility Function method

is as follows: a) Define a set of evaluation criteria for the KESP. In general these evaluation criteria will come from two sources: the user (functional and usability requirements) and organization (requirements such as low cost); b) Assign each evaluation criteria a weight (between 0-1) according to its importance—the weights of all the evaluation criteria must sum to 1. The objectives tree method [Pahl and Beitz, 1996] is a useful tool for assigning weights to the evaluation criteria; c) Score each design concept according to how well it satisfies each evaluation criteria using a convenient scale, e.g. 1-10; d) Calculate the overall utility U_i for each design concept i using Equation 6.1

$$U_i = \sum_{n}^{j=1} w_j * s_{ij}$$

,

where w_j denotes the weight of the $j-th$ evaluation criteria and s_{ij} denotes the score of design concept i with respect to the $j-th$ evaluation criteria.

5. Use the sub-functions from Function Structure to create the UML Component diagram for the KESP software architecture. Use the solution principles from the selected design concept to create the UML Class diagrams for the detailed software design. These outputs from the *Internal* layer are indicated by connections (6) and (7) in Figure 6.1.

The application of the *Internal* layer process to the development of the Service Request Portal resulted in the Function diagram shown in Figure 6.6, the Morphological Matrix shown in Figure 6.7, and the Utility Function shown in Figure 6.8. The Function Structure connects the NKEs' functional requirements to the SRP functional specification in order to ensure that the SRP satisfied requirements *R1a* and *R1b*. The Morphological Matrix and Utility Function enabled a wide range of possible designs for SRP to be explored and, therefore, maximized the probability that the SRP would be

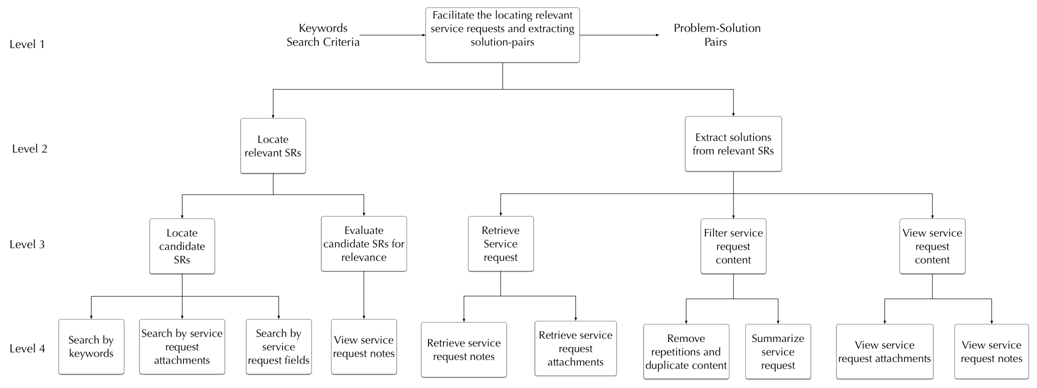

Level 1
Keywords Search Criteria
Facilitate the locating relevant service requests and extracting solution-pairs
Problem-Solution Pairs
Level 2
Locate relevant SRs
Extract solutions from relevant SRs
Level 3
Locate candidate SRs
Evaluate candidate SRs for relevance
Retrieve Service request
Filter service request content
View service request content
Level 4
Search by keywords
Search by service request attachments
Search by service request fields
View service request notes
Retrieve service request notes
Retrieve service request attachments
Remove repetitions and duplicate content
Summarize service request
View service request attachments
View service request notes

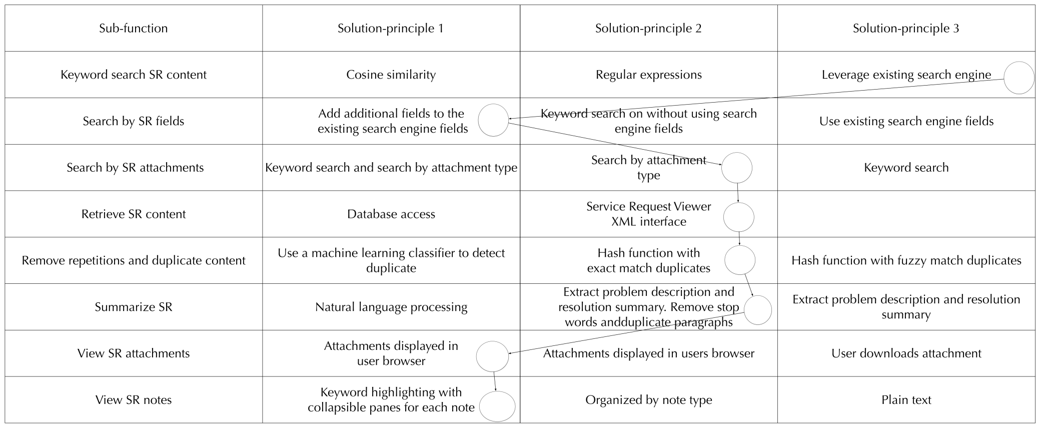
Figure 6.6: Function Structure for the Service Request Portal

high-quality (requirements *R2a* and *R2b*) and low cost (requirement *R3*).

Figure 6.6 shows the Function Structure of the key SRP functions and sub-functions. The Function Structure consisted of four levels, each representing an increasing detailed functional specification for the SRP. The first three levels followed directly from the primary and secondary functional requirements in the House of Quality (shown in Figure 6.4). The fourth level of the Function Structure decomposed the secondary functional requirements into the detailed product sub-functions which enabled the exploration of solution principles using the Morphological Matrix shown in Figure 6.7.

Figure 6.7 shows the Morphological Matrix that was used to generate three design concepts for the SRP. Design concept 1 was a high complexity design that used information retrieval methods such as “cosine similarity” and “natural language process” as solution principles. Design concept 2 was a low complexity design that used easy to implement solution-principles such as regular expressions. Design concept 3 (shown in Figure 6.7) used a “hybrid” approach that combined the existing search engine and service request viewer with the relatively simple solution principles from design concept 2.

Figure 6.8 compares the utility of the three design concepts with respect to the quality and cost evaluation criteria for the SRP. Examination of the Utility Function



Sub-function	Solution-principle 1	Solution-principle 2	Solution-principle 3
Keyword search SR content	Cosine similarity	Regular expressions	Leverage existing search engine
Search by SR fields	Add additional fields to the existing search engine fields	Keyword search on without using search engine fields	Use existing search engine fields
Search by SR attachments	Keyword search and search by attachment type	Search by attachment type	Keyword search
Retrieve SR content	Database access	Service Request Viewer XML interface	
Remove repetitions and duplicate content	Use a machine learning classifier to detect duplicate	Hash function with exact match duplicates	Hash function with fuzzy match duplicates
Summarize SR	Natural language processing	Extract problem description and resolution summary. Remove stop words andduplicate paragraphs	Extract problem description and resolution summary
View SR attachments	Attachments displayed in user browser	Attachments displayed in users browser	User downloads attachment
View SR notes	Keyword highlighting with collapsible panes for each note	Organized by note type	Plain text

Figure 6.7: Morphological Matrix and the selected design concept for the Service Request Portal

shows that Design Concept 3 yielded the highest utility (5.9) of the three concepts. The high utility score of design concept 3 was largely due to use of the existing search engine and service request viewer which provided the desired functionality at low cost. Design concept 1 scored highly with respect to the "quality" evaluation criteria but low on the "cost" criteria. Design concept 2 scored low with respect to the "quality" evaluation criteria but high on the "cost evaluation criteria.

6.5.4 Inside-Out Layer: Software Design of the Product

The *Inside-Out* layer translates the design concept and associated solution-principles from the *Internal* layer into the necessary software engineering artifacts for implementing the Knowledge Engineering Software Product (KESP) at the *Outer* layer.

The *Inside-Out* layer provides the link between the product design and software engineering methods that ensures that value created at the *Internal* layer is accurately transferred to the final product. To this end, the *Inside-Out* layer draws upon two tools from the Unified Modeling Language [Schach, 2008]. First, the UML Component diagram is used to translate the Function Structure into the software architecture and the overall structure for the KESP software implementation. Next, the UML Class

Evaluation Criteria		Weight	Concept 1		Concept 2		Concept 3	
			Score	Utility	Score	Utility	Score	Utility
Quality	Provide broad search capabilities	0.042	8	0.336	4	0.168	7	0.294
	Provide targeted search capabilities	0.126	6	0.756	3	0.378	7	0.882
	Be able to quickly assess the relevance of an SR	0.21	7	1.47	4	0.84	6	1.26
	Easy access to attachments	0.044	4	0.176	2	0.088	7	0.308
	Easy to find problem in service request	0.072	7	0.504	3	0.216	5	0.36
	Easy to find resolution in service request	0.072	7	0.504	3	0.216	5	0.36
Cost	Fast development time	0.28	3	O.84	8	2.24	6	1.68
	Low cost	0.12	3	0.36	4	0.48	7	0.84
Cumulative Utility		1		4.946		4.626		5.984

Figure 6.8: Utility Function used to select a high-value design concept for the Service Request Portal

Diagram is used to create the detailed design for implementing the solution-principles of the design concept.

The *Inside-Out* layer process for applying the Component and Class diagrams within the context of KESP development is as follows:

1. Group the terminal (bottom-level) sub-functions in the Function Structure so that each group is as self-contained as possible with minimal coupling to other groups. The number of groups will depend on the size and complexity of the design concept but will typically range from five to ten groups. This *Internal* layer input to the *Inside-Out* layer is indicated by connection (5) in Figure 6.1.

2. Create **UML Component** diagram [Schach, 2008] to define the software architecture of the KESP. The Component diagram for the Service Request Portal is shown in Figure 6.9. The process for creating this diagram is as follows: a) Create a software component for each sub-function group from Step 1. Label components that involve user interactions as UI components and components that only interact with other components as functional components; b) Add an actor for each distinct actor in the Use Case diagrams; c) Add the interactions between the actors and UI components based on the sub-function information flows in Function Structure and the Use Case diagrams. Add the interactions between UI components and functional components based on the sub-function information flows in the Function Structure.

3. Use a **UML Class diagram** [Schach, 2008] to decompose each component into a set of software classes. Figure 6.10 shows the Class diagram corresponding to the ContentFilter component from the Component diagram shown in Figure 6.9. The process for creating the Class diagram is as follows: a) Add an entity (data structure) class for each distinct input and output to the component; b) Add one or more functional classes to transform the input data structures into the output

data structures based on the solution-principles from the design concept; c) Add one or more control classes for handling the overall logic of the Component, e.g. receiving input data structures, triggering functional classes, and sending output data structures; d) Specify the relationships (inheritance, aggregation, association) between the entity, functional, and control classes.

4. Use the Component and Class diagrams at the *Outer* layer to develop the KESP. These two outputs from the *Inside-Out* layer are indicated by connections (8) and (9) in Figure 6.1.

The application of the *Inside-Out* layer process to the development of the Service Request Portal (SRP) resulted in the UML Component diagram shown in Figure 6.9 and six UML Class diagrams, one of which is shown in Figure 6.10. The Component and Class diagrams ensured that the value in the selected design concept was carried into the software design of the SRP. The Component diagram defined the two software components that the SRP would need to have in order to interface with the search engine and service request viewer. The Class diagrams provided the design for the data structures, functions, and control logic necessary to implement these two software components.

Figure 6.9 shows the six components in the software architectures of the Service Request Portal. The architecture consisted of three layers: user interface, search and content filtering, and back-end interface. The user interface layer contained the *SearchUserInterface* and *ContentUserInterface* components that handled the user interactions related to the "locate relevant service requests" and "extract solutions from relevant service requests" sub-functions. The *SearchFilter* and *ContentFilter* components handled the "locate candidate service requests" and "filter service request content" sub-functions. The back-end interface layer contained the *SearchEngineInterface* and *ServiceRequestViewerInterface* components that communicated with the existing search

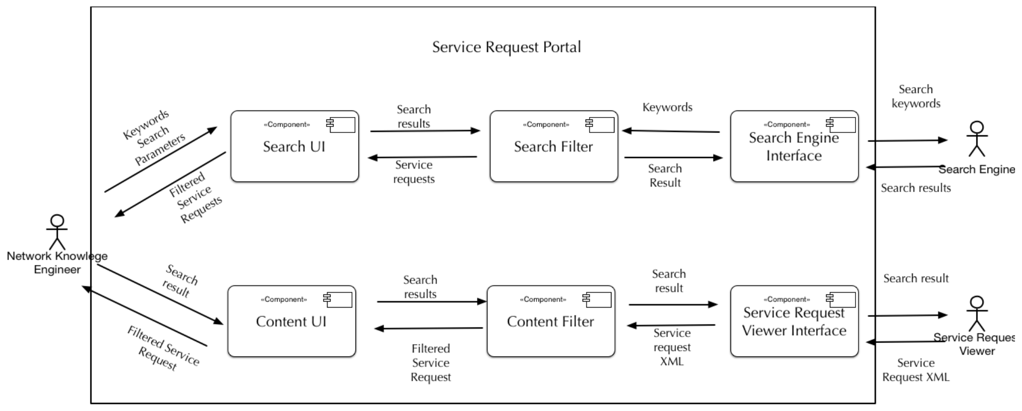


Figure 6.9: UML Component diagram showing the Service Request Portal software architecture

engine and service request viewer and handled the "locate candidate service requests" and "retrieve service request" sub-functions.

Figure 6.10 shows the Class diagram corresponding to the *ContentFilter* component. The *ContentFilter* component takes a set of search results as input and returns a set of filtered service requests as output. The transformation of search results into filtered service requests involved six classes. Representing the information flow for this component required two data structures: the *SearchResult* class to represent the search results and *ServiceRequest* class to represent the service requests. The hash-based deduplication solution-principle for transforming the search results into filtered service requests required two functional classes: the *DeduplicationFilter* class for removing duplicated content and *SummarizationFilter* for summarizing service requests. Lastly, the *ContentFilter* and *Filter* classes specified the control logic for managing the retrieval and filtering of the service requests.

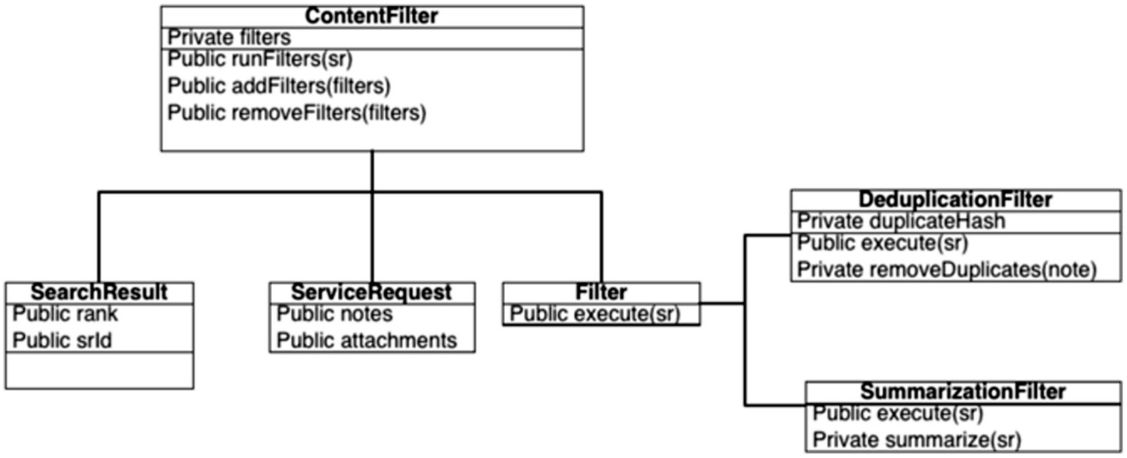


Figure 6.10: UML Class diagram showing the data structure, functional, and control logic classes for the Service Request Portal ContentFilter component

6.5.5 Outer Layer: Developing the Knowledge Engineering Software Product

The *Outer* layer addresses the planning, software development, and testing involved in transforming the software design from the *Inside-Out* layer into the finished Knowledge Engineering Software Product (KESP).

The *Outer* layer plays a critical role in ensuring the KESP satisfies all three requirements for high-value. For this reason, the *Outer* layer is based on an Iterative and Incremental Development (IID) [McConnell, 1996] life-cycle model that incorporates user feedback and testing through the use of short build and test cycles (iterations). Rapid build and test cycles with user feedback ensure that KESP is high-quality with respect to the user needs (requirements *R1a* and *R1b*) and minimize development time (requirement *R3*).

The *Outer* layer process for using Iterative and Incremental Development to develop the KESP is as follows:

1. Translate the Component diagram from the *Outside-In* layer into a series of soft-

ware development tasks for the KESP. Simple components will correspond to a single task, while more complex components will need to be broken up into several tasks. This input to the *Outer* layer is indicated by connection (7) in Figure 6.1.

2. Create a software development plan [Schach, 2008] for implementing the KESP. The software development plan is created as follows: a) Organize the development tasks from Step 1 into a series functional (user testable) prototypes; b) Determine the relationship between the prototypes and the user needs specified in the House of Quality; c) Use the House of Quality to prioritize the prototypes so that the functionality related to the important user needs is addressed first.

3. Prototype the KESP using **Iterative and Incremental Development** (IID) [McConnell, 1996]. The process for applying IID is as follows: a) Use the Component diagram to develop the interfaces between components; b) Use the Class diagrams to develop the classes. These inputs to the prototyping process are indicated by connections (8) and (9) in Figure 6.1.

4. Test the prototype with users and elicit feedback. The process for testing the prototype is as follows: a) Have the users actively use the prototype in their work process (captured in the Agent/Task model); b) Elicit feedback for each user need using a convenient scale (e.g. exceeds need, meets need, does not meet need). Record measurements for each technical metric in the House of Quality; c) Review the user feedback and technical metrics for the current prototype. If the user feedback and technical metrics are satisfactory (based on the House of Quality) then the prototype is finished, and development proceeds to the next increment in the software development plan. Perform another iteration if the prototype is significantly below user expectations or the targets for the technical metrics.

5. Deploy the KESP to the end-users. This output from the *Outer* layer is indicated by connection (9) in Figure 6.1) and represents satisfaction of the goal state of

Prototype	Components	Development Tasks
Alpha	*SearchUserInterface* *SearchFilter* *SearchEngineInterface*	*SearchUserInterface* *SearchFilter* *SearchEngineInterface*
Beta	Content UI *ContentFilter* *SearchEngineInterface*	*ContentUserInterface* *ContentFilter*
Release Candidate	Search UI *SearchFilter* *ContentUserInterface* *ContentFilter* *SearchEngineInterface*	Integration

Table 6.3: software development plan for implementing the Service Request Portal

the IRPM.

The *Outer* layer of the Service Request Portal involved three increments (prototypes) shown in Figure 6.3: alpha, beta, and release candidate. The alpha prototype implemented the search filter functionality (Search UI, Search Filter, and Search Engine Interface). The beta prototype implemented content filter functionality (Content UI, Content Filter, and Search Engine Interface). The release candidate prototype, integrated the search filter and content filter into a complete KESP. The Incremental and Iterative Development process implemented the value generated at the *External*, *Outside-In*, and *Internal* layers. The *Outer* layer produces the final SRP software product described at the beginning of Section 6.5. The evaluation of the SRP with respect to the organization needs, NKE needs, and value requirements is described in Section 6.7.

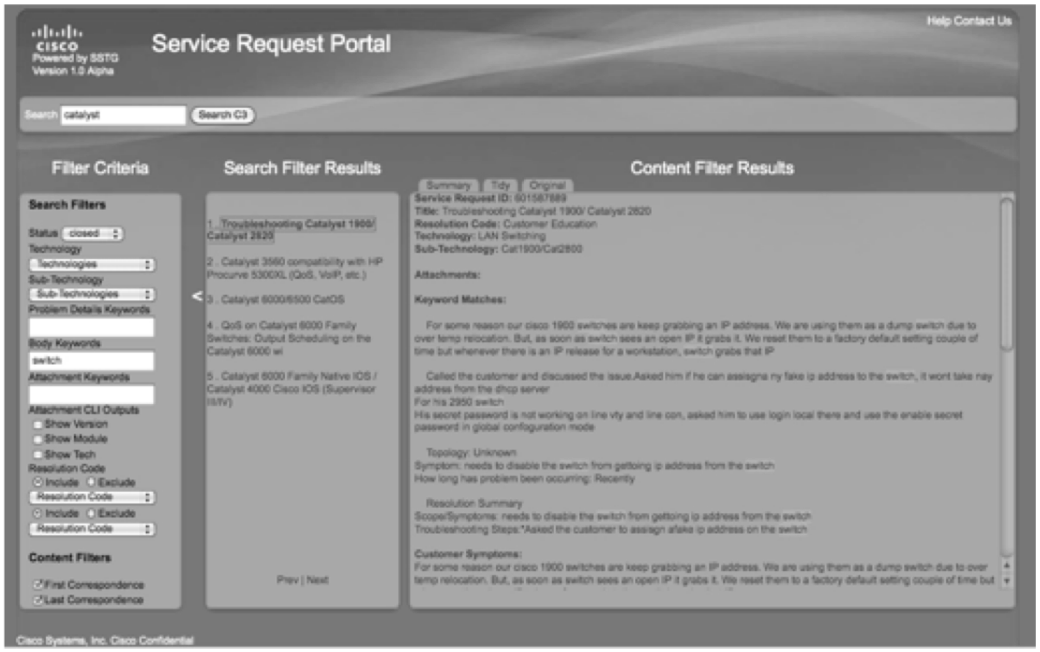
6.6 Software Environment

The Service Request Portal (SRP) provides search and content filtering to automate the manual information retrieval activities in the Network Knowledge Engineers' (NKEs) work process for extracting problem-solution pairs from service requests. The NKEs interact with the SRP using the web-based graphical user interface (GUI) shown in

Figure 6.11. The NKEs first enters a set of search keywords in the text box in the top pane of the GUI. The search results are then displayed in the middle pane of the GUI. Next, the NKEs refine the search results by adding additional search criteria (e.g. "device hardware type") that act as constraints on the results. Once irrelevant search results are removed, the NKEs can open the relevant service requests by clicking on the search result. The corresponding service request is displayed in the right pane of the GUI. The NKEs are then able to select from three different service request "views", each providing a different level of information. The first view is a high-level summary that captures the essential information from the service request—problem experienced by the customer, the first and last correspondence between the customer and technical support engineer, and the resolution summary of the steps taken to solve the problem. The second view is a complete but deduplicated and reformatted version of the service request that is color coded in order to simplify reading. The third view is the unedited service request taken directly from the service request viewer.

The SRP software architecture, shown in Figure 6.12, consists of four major blocks: the SRP application, the existing search engine that was used by the NKEs to locate service requests, the existing service request viewer used by the NKEs to read service requests, and the relational database where the service requests are stored. The SRP application combines the GUI shown in Figure 6.11 with search and content filtering functionality layered on top of the existing search engine and service request viewer. The search filtering functionality uses the results from the search engine and applies additional criteria, provided by the NKEs, to filter out irrelevant results. The content filtering functionality uses the service request viewer to retrieve the service request and applies a set of filters to remove duplicated information and provide a concise summary of the service request.

The SRP tool provided two key features to support the SCH engineers work process: domain specific search features that made it easier that made it easier to locate relevant



Search: catalyst

Search Filters

Status: closed
Technology: Technologies
Sub-Technology: Sub-Technologies
Problem Details Keywords:
Body Keywords: switch
Attachment Keywords:
Attachment CLI Outputs
- ☐ Show Version
- ☐ Show Module
- ☐ Show Tech

Resolution Code
[✓ Include] Exclude
Resolution Code
[✓ Include] Exclude
Resolution Code

Content Filters

- ✓ First Correspondence
- ✓ Last Correspondence

1 . Troubleshooting Catalyst 1900/ Catalyst 2820

2 . Catalyst 3560 compatibility with HP Procurve 5300XL (QoS, VoIP, etc.)

3 . Catalyst 6000/6500 CatOS

4 . QoS on Catalyst 6000 Family Switches: Output Scheduling on the Catalyst 6000 wi

5 . Catalyst 6000 Family Native IOS / Catalyst 4000 Cisco IOS (Supervisor III/IV)

Service Request ID: 601587689
Title: Troubleshooting Catalyst 1900/ Catalyst 2820
Resolution Code: Customer Education
Technology: LAN Switching
Sub-Technology: Cat1900/Cat2800

Attachments:

Keyword Matches:

For some reason our cisco 1900 switches are keep grabbing an IP address. We are using them as a dump switch due to over temp relocation. But, as soon as switch sees an open IP it grabs it. We reset them to a factory default setting couple of time but whenever there is an IP release for a workstation, switch grabs that IP

Called the customer and discussed the issue.Asked him if he can assisgna ny fake ip address to the switch, it wont take nay address from the dhcp server
For his 2950 switch
His secret password is not working on line vty and line con, asked him to use login local there and use the enable secret password in global confoguration mode

Topology: Unknown
Symptom: needs to disable the switch from gettoing ip address from the switch
How long has problem been occurring: Recently

Resolution Summary
Scope/Symptoms: needs to disable the switch from gettoing ip address from the switch
Troubleshooting Steps:"Asked the customer to assisgn afake ip address on the switch

Customer Symptoms:
For some reason our cisco 1900 switches are keep grabbing an IP address. We are using them as a dump switch due to over temp relocation. But, as soon as switch sees an open IP it grabs it. We reset them to a factory default setting couple of time but

Figure 6.11: Graphical user interface for the Service Request Portal

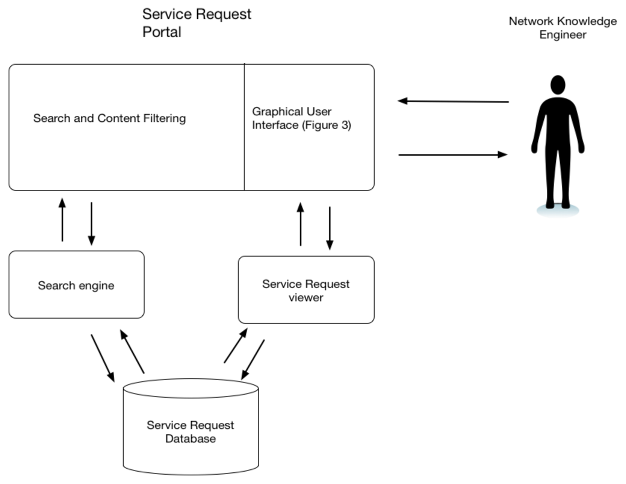

Service Request Portal
Network Knowledge Engineer
Search and Content Filtering
Graphical User Interface (Figure 3)
Search engine
Service Request viewer
Service Request Database

Figure 6.12: Software architecture for the Service Request Portal

service and automatic summarization of long and complicated service requests. The SRP was implemented as a client server architecture consisting of three pieces: a Ruby backend that handled the retrieval and summarization of service requests, a MySQL database for storing service requests, and a HTML/Javascript web interface that the SCH engineers connected to using a web browser.

6.7 Results: Knowledge Engineering Software Product for

In this section we present key results from our evaluation the Service Request Portal (SRP) with respect to the three requirements for high-value that were developed in Section 6.2.

6.7.1 User Needs

The first requirement for high-value is that the Knowledge Engineering Software Product (KESP) must be high-quality with respect to the user needs. This requirement has two parts: *R1a* and *R1b*. Requirement *R1a* addresses the need for integration with the knowledge engineers' existing work process, while requirement *R1b* addresses the user experience with the KESP.

In order to determine how well the SRP satisfied requirements *R1a* and *R1b* we collected user feedback for the functional and usability requirements in the House of Quality during each build and test cycle. The Network Knowledge Engineers (NKEs) provided feedback on the following three point scale: "Does not meet need", "Met need", or "Exceeded need".

Table 6.4 shows the results from testing the SRP with the NKEs. The NKEs wanted a software product to automate the tedious and manual aspects of their work process related to the "Breadth", "Depth", "Summary", and "Repetition" problems. The SRP

User Need	Evaluation	Feedback
“Be able to do broad searches”	Exceeded need	“The Service Request Portal helps us find information easily compared to the current search engine”
“Be able to do very targeted searches”	Met need	
“Be able to read service requests easier”	Exceeded need	
“Be able to quickly extract problem-solution pairs”	Met need	“The Service Request Portal is useful to extract the information quickly”
“Seamless access to service request attachments”	Met need	
“Easy to use”	Exceeded need	“The Service Request Portal looks just amazing. Our team is excited to use it for rule writing.”
“High performance”	Exceeded need	

Table 6.4: User evaluation of the service request portal

addressed the “Breadth”, “Depth” problems by allowing the NKEs to specify granular search filters when searching for service requests. The SRP addressed the “Summary”, and “Repetition” problems by providing the NKEs with a high-level summary for each service request and a deduplication filter that removed the most of the irrelevant content in long and complicated service requests. Since the SRP met or exceeded the functional and usability requirements shown in Table 6.4, we conclude that the SRP satisfied requirements *R1a* and *R1b* for a high-value KESP.

6.7.2 Productivity Improvement

The second requirement for a high-value Knowledge Engineering Software Product (KESP) is that the KESP must improve the knowledge engineering processes. This requirement has two parts: *R2a* and *R2b*. Requirement *R2a* addresses the difference the SRP made with respect to the NKEs' work process, while requirement *R2b* addresses the impact of the SRP with respect to the overall organizational goal to develop smart networking products and services. We will focus on evaluating the SRP with respect to requirement *R2a* because it can be measured directly through user testing.

In order to determine how well the Service Request Portal satisfied *R2a* we evaluated the productivity impact of the SRP on several key aspects of the Network Knowledge Engineers' (NKEs) work process. The NKEs were split into two groups and each group was given the task of creating problem-solution pairs for nine product problems that ranged from a relatively simple voltage alarm (R3) to complex hardware interface problem (R1). The first group used the SRP to locate relevant service requests and extract problem-solution pairs. The second group used the existing search engine and service request viewer. The productivity of each group was benchmarked using the following three technical metrics from the House of Quality in Figure 6.4.

1. **TM1**: "average number of pages read to assess relevance of a service request"
2. **TM2**: "average time spent assessing relevance of a service request"
3. **TM3**: "average time to extract the problem-solution pairs from the relevant service requests"

The SRP reduced the "average number of page read to assess relevance of a service request" (TIM1) from 22 pages to 3 pages (Figure 6.13). The NKEs attributed the decrease in number of pages read to the service request summary. By reducing the

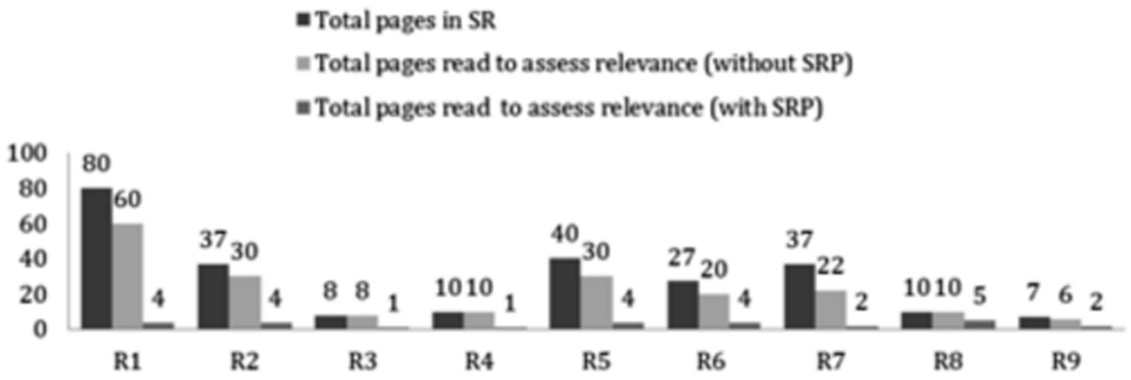


Figure 6.13: Technical Metric 1: Average number of pages read to assess relevance of a service request

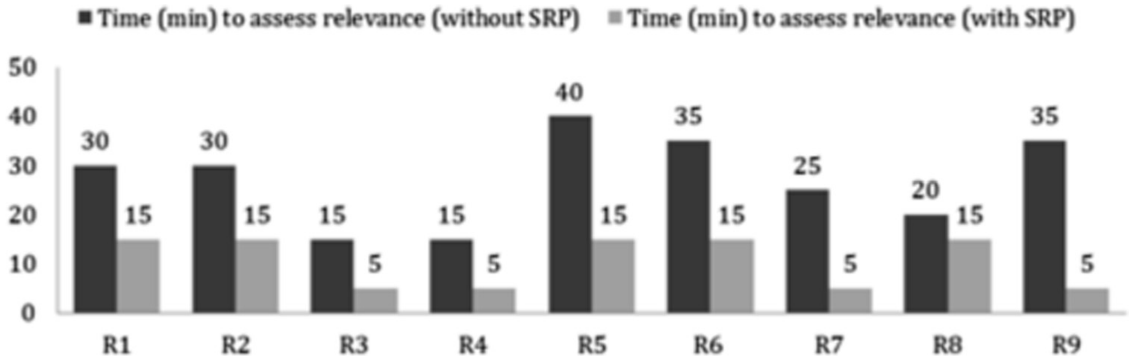


Figure 6.14: Technical Metric 2: Average number of pages read to assess relevance of a service request

number of pages the NKEs had to read the SRP also reduced the "average time spent assessing relevance of a service request" (TIM2) by 60% (see Figure 6.14).

Once the NKE's identified a relevant set of service requests, the next step involved extracting the problem-solution pairs from the relevant service requests. Figure 6.15 shows that the SRP reduced the "average time to extract the problem-solution pairs from the relevant service requests" (TIM3) by over 30%. The time savings were largely from the content filter that removed duplicate information from the service requests and user interface features such as keyword highlighting.

The networking company's problem statement was to improve the productivity of the NKEs and decrease the cost of developing problem-solution pairs. The results for

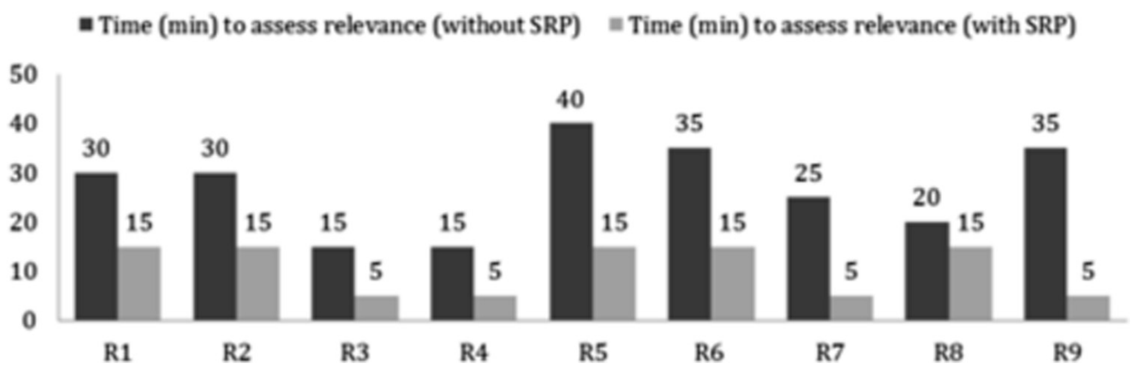


Figure 6.15: Technical Metric 3: Average time to extract problem-solution pair from a service request

technical metrics TM1, TM2, and TM3 demonstrate that the SRP made a significant improvement to NKEs' productivity and, therefore, satisfies requirement *R2a*. We were not able to directly evaluate the impact of the SRP on the cost of extracting problem-solution pairs (requirement *R2b*), however, the results for requirement *R2a* suggest that the SRP will also significantly decrease the costs involved.

6.7.3 Cost Effectiveness

The third and final requirement for high-value is that "the Knowledge Engineering Software Product must be developed rapidly and at low cost" (requirement *R3*). In order to evaluate the SRP with respect to this requirement we compared the projected time-savings of the NKEs using the SRP (based on the results for requirements *R2a* and *R2b*) to the SRP development costs (time) and assessed how long the SRP will take to produce a positive return on investment (ROI).

The design and development of the SRP involved approximately 1000 hours of work carried out by two software engineers over a six-month period. The resulting product, the SRP, was used by a team of 10 Network Knowledge Engineers (NKEs) as the primary tool for supporting their daily work process. On a typical day the NKEs read 10 service requests for relevance and extracted problem-solution pairs from two of the relevant service requests. Based on the increased productivity described in the previous section (15 minute when assessing relevance and 30 minutes when extracting problem-solution pairs), we estimate that the daily productivity of the NKEs would increase by 3.5 hours when using the SRP. For the team of 10 NKEs, the productivity impact of using the SRP will be on the order of 35 hours every day. Assuming that one hour of the development engineers' time is equal in value to one hour of the NKEs' time, the SRP will produce a positive return on investment (ROI) after one and a half months (43 days) after it is deployed to the NKEs.

In order to achieve the networking company's overall objective, decrease the cost of involved with extracting problem-solution pairs, the SRP also had to be cost-effective. The SRP relatively low development cost (43 days until a positive ROI) satisfies requirement *R3* for high-value.

6.7.4 Simplifications

The Integrated Representation Based Process Methodology (IRPM) is a comprehensive methodology for developing high-value Knowledge Engineering Software Products (KESPs). We recommend that the complete IRPM be used whenever possible, however, there are conditions where it is necessary to simplify the IRPM in order to increase the overall value of the KESP. This section describes the potential simplifications to the IRPM and provides the guidelines for how these simplifications should be implemented. The Service Request Portal (SRP) from Section 6.5 is used as an illustrative example for applying the simplifications.

There are three different conditions for simplifying the IRPM. First, a time constraint might limit the amount of time that can be spent for each layer of the IRPM. Second, a resource constraint might limit the number of people, money, and/or computer hardware available for developing the KESP. Third, the problem under consideration might be of low complexity and not significantly benefit from some of the formal tools in the IRPM. Table 6.5 shows the recommended simplifications to the *External*, *Outside-In*, *Internal*, and *Outer* layers of IRPM.

The process for applying the simplifications listed in Table 6.5 is as follows:

1. Identify the constraints for the KESP development process. Typically there will be three different types of constraints: time constraints, human resource constraints, and financial constraints.
2. Determine the key characteristics of the problem under consideration. Important characteristics to consider include: size of the organization, number of end-users for the KESP, complexity of end-user's work process, and importance of the problem to the organization.
3. Select the appropriate set of simplifications from Table 6.5 based on the results from Step 2 and apply them when using the IRPM to develop the KESP.

Note that Table 6.5 does not recommend simplifying the *Inside-Out* layer of the IRPM (software design). This layer is critical in successfully implementing the design concept from the *Internal* layer at the *Outer* layer and its simplification could result in the KESP not meeting the three requirements for high-value.

The development of the SRP, described in Section 6.5, had both time and resource constraints—the product had to be developed in six months by two software engineers—that required the simplification of the *External*, *Outside-In*, and *Internal* layers of the IRPM. At the *External* layer we interviewed stakeholders—end-users and their managers—in order to capture the organizational context and end-user work processes for the KESP but did not create the CommonKADS Organization or Agent/Task models. At the *Outside-In* layer we defined the user needs and technical metrics for evaluating the KESP but did not formally correlate them in a House of Quality. At

Layer	Method	Conditions for Simplification	Simplifications	Estimated Time Savings	Impact (Consequences)
External	Organizational Model	The KESP is being developed within a smaller organization and/or the Knowledge Engineering problem under consideration is straightforward	List the people and processes without constructing the Organization model diagram	Medium to High	Lower probability of the KESP development being focused on the right problem.
	Agent/Task Model	List tasks without creating the Agent/Task diagram	The end-users' current work-process is relatively simple	Medium to High	Might miss important relationships between tasks
Outside-In	House of Quality	There are a small number of user needs and/or the relationship between the user needs and technical metrics is straightforward	List the user needs and technical metrics without creating the relationship and correlation matrices	Low to Medium	More difficult to make trade-offs during the development process.
	Use Case Diagrams	The end-user work process is relatively simple	List the steps in the Use Case but do not create a UML diagram	Low - Medium	More difficult to integrate the KESP into the end-user work process
Internal	Function Structure	Relatively simple functional user needs and use cases	Reduce depth of the Function Structure and omit information flows	Low - Medium	More difficult to come up with solution-principles when creating the Morphological Matrix
	Morphological Matrix	Sub-functions are relatively simple and straightforward to implement	Reduce the number of solution-principles explored for each sub-function	Low to Medium	Lower probability of generating the best design with respect to the user and organizational needs
	Utility Function	Small number of evaluation criteria (5-10)	Do an informal comparison of the of the design concepts using the evaluation criteria	Low to Medium	Reduced objectivity during design selection
Outside-In	Component Diagram	Simplification is not recommended			
	Class Diagrams	Simplification is not recommended			
Outer	Software Development Plan	Simplification is not recommended			
	Iterative and Incremental Development	Quality (robustness, ease of use, etc.) is not critical	Reduce the number of iterations	Medium to High	Less user feedback

Table 6.5: Possible simplifications for each layer of the process methodology for knowledge engineering software product development

the *Internal* layer we developed the functional specification for the SRP and explored multiple design concepts but did not use the formal methods (Function Structure, Morphological Matrix, and Utility Function). The *Inside-Out* and *Outer* layer processes were not simplified. Without these simplifications the development of SRP would have required significantly more resources to develop.

It is important to note that a number of the SRP examples for the methods in Section 6.5—in particular the Organization model, Agent/Task model, House of Quality, Function Structure, Morphological Matrix, and Utility Function—were not created during the development of the SRP. These examples were created after the SRP had already been developed in order to illustrate the application of the IRPM within the context of a real knowledge engineering problem.

6.8 Conclusions

Rapid and cost effective knowledge engineering requires the automation through the creation of Knowledge Engineering Software Products (KESPs). However, the deployment of KESPs often does not yield useful results, in particular because insufficient attention is spent addressing the needs of users that will be using the system. In this application we have used the Integrated Meta-Representation Model to combine methods and techniques from the domains of Knowledge Engineering, Engineering Design, and Software Engineering into a unified process methodology for developing KESPs.

Our novel contribution is the use of methods and techniques from Engineering Design in order to resolve the typical trade-off conflicts that arise during design, development, and deployment and ensure that the developed product sufficiently addresses the users needs and is high-value (quality and cost). We have demonstrated the effectiveness of the process methodology within the context of a simple but non-trivial KESP for supporting the development of smart services in the computer-networking domain.

7 Conclusion and Future Research

In this concluding chapter we first summarize the research presented in the thesis. We then discuss the contributions from the research. Lastly, we lay out future research directions that addresses the generalization of the research contributions.

7.1 Summary of the Thesis

There are an increasing number of organizations attempting to apply knowledge engineering to support the transformation of massive amounts of data and information into useful knowledge that can be used to influence core business activities, e.g. product development, customer support, and marketing. Recent advances in distributed computing, storage systems, and machine learning have provided analytic tools and databases that can handle the processing extremely large volumes of data. Despite these advances, the extraction of knowledge, which can then be used for engineering applications, e.g. the design, development, and support of engineering systems and products, is still a major challenge for many organizations.

In order to address this issue we have developed a theoretical framework consisting of an Integrated Meta-Representation Model (IMRM) for structuring complex knowledge engineering problems, and then applied the IMRM to create representation-based models for addressing three important issues that arise in the application of knowledge engineering: domain knowledge modeling, machine learning model selection, and the development of knowledge engineering software products. The structure provided by the IMRM enables the appropriate subject matter domains and associated methods and tools to be identified for each particular problem.

We have demonstrated this framework using three knowledge engineering problems at a large enterprise that designs, develops, and delivers (supports) computer networking products and services. The developed representation-based models integrate methods and tools from four diverse subject matter to address the three important knowledge engineering problems. To address the domain knowledge modeling issue, described in Chapter 4, we used engineering design methods to model domain knowledge that allowed us to translate unstructured customer problem descriptions into engineering failure modes that could then be used to monitor and assess the quality of a computer network security device. In the service request escalation problem, described in Chapter 5, we used of methods from statistical design of experiments in order to efficiently select the machine learning model that optimized the objective of minimizing total escalation cost. In the development of knowledge engineering software products problem, described in Chapter 6, we used engineering design methods to develop a high-value knowledge engineering software product to improve the productivity of engineers extracting problem-solution pairs from customer service requests.

The IMRM should prove to be valuable to the increasing number of organizations attempting to transform massive amounts of unstructured data and information into useful knowledge that can be used to influence core business and technology activities, e.g. product development, customer support, and marketing. When addressing problems involving engineering applications, e.g. product design, the desired output or result of the knowledge engineering process is often unknown. For example, consider the problem discussed earlier where the computer networking organization wanted to monitor the quality of products in service. The organization knew that it wanted to use customer service requests to monitor quality, however, the details of how quality would be monitored were unknown.

The use of representation-based methods enabled the issues associated with this problem to be layered so that we can identify the different representations of the problem (*External*, *Internal*, and *Outer* layers) and the intermediate representations for transitioning between these problem representations (*Outside-In* and *Inside-Out* layers).

The proper layering of the complex problem to be solved then enables and facilitates the difficult task of identifying the appropriate subject matter domains as well as the appropriate methods and tools in these domains.

7.2 Thesis Contributions

In a seminal essay, "No Silver Bullet" [Brooks, 1987], Brooks made the argument that complexity in software engineering problems can be organized into "essential" complexity that is inherent to the problem under consideration and "accidental" complexity that is an artifact of the tools that are available to software engineers. He goes on to assert that the most of the big productivity gains in software engineering have come from progress on the accidental complexity component through better tools (higher-level programming languages, faster computers, etc.) and that future productivity improvements must come from addressing the essential complexity.

We can argue that we are at a similar point for knowledge engineering as a discipline. In the past decade the knowledge engineering tool-set has evolved, and as a result practitioners have increasingly better algorithms and computer systems at their disposal. These tools remove much of the accidental complexity of knowledge engineering by simplifying the process of manipulating large data-sets, developing machine learning models, and deploying software systems. However, these tools do not ease the essential complexity involved in knowledge engineering—the cognitive burden associated with large scale problems spanning multiple engineering and business domains. For example, new machine learning algorithms enable us to rapidly build extremely accurate predictive models from large data-sets. However, these algorithms do not help the knowledge engineer determine the organizational context, user needs, and business and technical objectives for the model.

The novel contribution of this work is the notion of representation-based models for structuring and subsequently solving complex knowledge engineering problems. The use of representation provides a structured approach to attacking the fundamental or essential complexity of knowledge engineering problems. The representation-based model approach, resulting from applying the IMRM to a particular problem, has the following features that address the essential complexity of knowledge engineering problems:

1. Each layer is a representation (or map) of the necessary steps in the process of designing and developing a high-value solution to solve a particular problem. The five layers of the IMRM are labelled as follows: *External*, *Outside-In*, *Internal*, *Inside-Out*, and *Outer*. The first layer, the *External*, represents the initial state of the problem under consideration; while the fifth layer, the *Outer*, represents the complete solution to the problem.

2. As one progresses through the five layers of representation, the level of abstraction first increases (*External* to *Internal*) and then decreases (*Internal* to *Outer*). By focusing the work at the appropriate level of abstraction, the IMRM allows for a more comprehensive approach to ensuring and maximizing the satisfaction of the customer needs and resolving trade-offs between quality and cost.

3. The sequential and functional layering of the IMRM supports the concurrent selection of the appropriate methods and tools and their placement in the proper layer. This enables the functional ("input-output") integration of the selected methods and tools, and thereby, facilitates a seamless transition between the layers.

We have demonstrated the application and implementation of the IMRM to the three complex multi-disciplinary problems summarized in the previous section.

7.3 Future Work

We have applied the Integrated Meta-Representational Model to three important knowledge engineering problems. In the process we have shown that the representation-based approach helps manage the complexity associated with solving multi-disciplinary problems that involve the four different key activities of engineering: design, analysis, experimentation, and manufacturing/prototyping. For each of the three problems we focused on one of these four engineering activities. This simplification enabled us to demonstrate the application of the IMRM to complex real-world knowledge engineering problem within the time and resource constraints of the thesis.

Our hypothesis, based on the experience of solving the aforementioned problems, is that the representation-based approach provided by the Integrated Meta-Representational Model (IMRM) becomes increasingly valuable as the size and complexity of the problem increases. The major research question to be answered with future work is testing this hypothesis using a large-scale knowledge engineering problem in which all four engineering activities are significant.

In order to illustrate a problem involving all four engineering activities consider the following scaled-up instance of the quality monitoring problem discussed in Chapter 4. The associated tasks for each of the four engineering activities are as follows:

- **Design**: Design a high-quality monitoring and assessment knowledge engineering software product (KESP) for the product design, development, and delivery teams to use.
- **Analysis**: Perform real-time analysis of service requests for quality monitoring and assessment.
- **Experiments**: Determine the optimal machine learning model for classifying service requests and the optimal forecasting model for time-series analysis.

- **Prototyping / Manufacturing**: Build and test the knowledge engineering software product.

Since this problem is essentially a super-set of the three problems addressed in this thesis, addressing these tasks requires methods and tools from all of the engineering domains considered in this thesis: machine learning, time-series analysis, knowledge engineering, statistical design of experiments, product design and development, and software engineering. The subject of future work would therefore be: 1) the application of the IMRM to appropriately layer the four engineering activities so that the necessary methods and tools from the relevant engineering domains can be rationally selected, and 2) the implementation of the resulting complete engineering representation-based model in order to create a comprehensive solution for the product quality monitoring and assessment problem. Addressing and resolving this problem would demonstrate that the Integrated Meta-Representational Model is ready to address complex and large-scale technical problems beyond the knowledge engineering problems addressed in this thesis.

References

C. Alexander. *Notes on the Synthesis of Form.* Harvard University Press, 1964.

J. Angele, D. Fensel, D. Landes, S. Neubert, and R. Studer. Model-based and incremental knowledge engineering: The mike approach. *AIFIPP*, 1992.

Croft. B., Metzler. D., and Stroham. T. *Search Engines: Information Retrieval in Practice.* Addison Wesley, 2009.

J. Bergstra, Bardenet R., Bengio Y., and B. Kgl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems (pp. 2546-2554)*, 2011.

J. Bergstra, Bardenet R., Bengio Y., and B. Kgl. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *ICML*, 2013.

J. Bermudez. *Cognitive science: An introduction to the science of the mind.* Cambridge University Press, 2014.

B. Boehm. Value-based software engineering. *ACM Software Engineering Notes*, 2003.

B. Bowerman, R. O'Connel, and A. Koehler. *Forecasting, Time Series, and Regression.* Cengage Learning, 2004.

R. Brachman and M. Levesque. *Knowledge Representation and Reasoning.* Morgan Kaufmann, 2004.

P.B. Brazdil, C. Soares, and J.P. Da Costa. Ranking learning algorithms: using ibl and meta-learning on accuracy and time results. *Mach. Learn. 50(3), 251-277.*, 2003.

L. Breiman. Bagging predictors. *Machine Learning 24, 123-140.*, 1996.

F. Brooks. No silver bullet essence and accidents of software engineering. *IEEE Computer*, 1987.

A. Card, J. Ward, and P. Clarkson. Beyond fmea: The structured whatif technique (swift). *Journal of Healthcare Risk Management*, 2012.

R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd international conference on Machine learning*, 2006.

S. Chopra. *Supply Chain Management, 4/e.* Pearson Education, 2007.

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning 20.3: 273-297*, 1995.

S. Desa and S. Kannapan. Value centered model of product design, development, and delivery. *ASME Design and Theory and Methodology Conference*, 1995.

S. Desa and T. Munger. A representation-based methodology for developing high-value knowledge engineering software products: Theory, application, and implementation. *ASME Journal of Computing and Information Science in Engineering*, 2013.

R.A. Fisher. *The Design of Experiments.* Oliver and Boyd, Edinburgh., 1935.

George Forman, Evan Kirshenbaum, and Jaap Suermondt. Pragmatic text mining: minimizing human effort to quantify many issues in call logs. *ACM Knowledge Discovery and Data Mining*, 2006.

W. Fowlkes and C. Creveling. *Engineering methods for robust product design.* Addison-Wesley., 1995.

J. Fox. *Quality Through Design: The Key to Successful Product Delivery.* Mcgraw-Hill, 1993.

G. Franklin, J. Powell, and A. Emami-Naeini. *Feedback control of dynamic systems.* Addison-Wesley, 1994.

J. Gama and P. Brazdil. Characterization of classification algorithms. *Progress in Artificial Intelligence*, 1995.

M. Garcia, M. Sanz-Bobi, and J. Pico. Simap: Intelligent system for predictive maintenance: Application to the health condition monitoring of a windturbine gearbox. *Computers in Industry 57.6*, 2006.

J. Ginsberg, MH. Mohebbi, RS. Patel, L. Brammer, MS. Smolinski, and L. Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 2009.

C. Glymour, D. Madigan, D. Pregibon, and P. Smyth. Statistical themes and lessons for data mining. *Data mining and Knowledge Discovery 1(1), 11-28.*, 1997.

M. Harry and R. Schroeder. Six sigma: the breakthrough management strategy revolutionizing the world's top corporations. *Broadway Business*, 2006.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition.* Springer, 2009.

J. Hauser and D. Clausing. The house of quality. *Harvard Business Review*, 1988.

V. Hubka, N. Andreasen, and E. Eder. *Practical Studies in Systematic Design.* Butterworth, London, 1988.

M. Jackson. *System Development.* Prentice Hall, 1983.

M. Jackson. *Problem Frames: Analyzing and structuring software development problems.* Addison Wesley, 2001.

K. Kotovsky, J. Hayes, and H. Simon. Why are some problems hard? evidence from tower of hanoi. *Cognitive psychology 17.2 248-294.*, 1985.

J. Lawson and J. Erjavec. *Modern Statistics for Engineering and Quality Improvement.* Duxbury, 2001.

W. Lee, D. Grosh, F. Tillman, and C. Lie. Fault tree analysis, methods, and applications: A review. *IEEE Transactions on Reliability*, 1985.

G. Lindner and R. Studer. Ast: Support for algorithm selection with a cbr approach. *Springer Berlin Heidelberg*, 1999.

J. Lombardo, H. Burkom, and J. Pavlin. Essence ii and the framework for evaluating syndromic surveillance systems. *Morbidity and Mortality Weekly Report*, 2004.

T. Lotze and G. Shmueli. How does improved forecasting benefit detection? an application to biosurveillance. *International Journal of Forecasting*, 2009.

T. Lotze and G. Shmulei. Ensemble forecasting for disease outbreak detection. *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, 2008.

S. McConnell. *Rapid Development: Taming Wild Software Schedules.* Microsoft Press, 1996.

A. Medem, M. Akodjenou, and R. Teixeira. Troubleminer: Mining network trouble tickets. *In Proc. of the 1st IFIP/IEEE international workshop on Management of the Future Internet*, 2009.

T. Metzinger. *Being No-One.* MIT Press, 2003.

D. Michie, D. Spiegelhalter, and C. Taylor. Machine learning, neural and statistical classification. *Ellis-Horwood*, 1994.

T. Munger. A representation-based methodology for developing high-value knowledge engineering systems: Theory and applications. Master's thesis, University of Califonia, Santa Cruz, 2012.

T. Munger, S. Desa, and C. Wong. The use of domain knowledge models for effective data mining of unstructured customer service data in engineering applications. *IEEE Big Data Services*, 2015.

K. Otto and K Wood. *Product Design: Techniques in Reverse Engineering and Product Design*. Prentice Hall, 2000.

G. Pahl and W. Beitz. *Engineering Design: A Systematic Approach*. Springer-Verlag, 1996.

B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Tell me who can learn you and i can tell you who you are: Landmarking various learning algorithms. *Proceedings of the 17th international conference on machine learning.*, 2000.

M. Phadke. *Quality Engineering Using Robust Design*. Prentice Hall, 1989.

R. Potharaju, N. Jain, and C. Nita-Rotaru. Juggling the jigsaw: Towards automated problem inference from network trouble tickets. *In Proceedings of Networked Systems Design and Implementation (NSDI)*, 2013.

S. Pugh. *Total Design*. Addison Wesley, 1991.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

A. Revonsuo. *Inner Presence: Consciousness as a Biological Phenomenon*. MIT Press, 2009.

J. Rice. The algorithm selection problem. *Advances in Computing 15(1): 65-118.*, 1976.

S.J. Russell, P. Norvig, J.F. Canny, J.M. Malik, and D.D. Edwards. *Artificial intelligence: a modern approach*. Prentice Hall, 2003.

S. Schach. *Object Oriented Software Engineering*. Prentice Hall, 2008.

R. Schapire. The strength of weak learnability. *Machine Learning 5, 197-227.*, 1990.

B Schreiber, G. Wielinga. Commonkads: a comprehensive methodology for kbs development. *IEEE Expert Systems*, 1994.

C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. *Data Mining and Knowledge Discovery 4(2-3), 163-192.*, 2000.

H. Simon. The structure of ill structured problems. *Artificial intelligence 4.3-4 181-201.*, 1973.

MR. Smith, L. Mitchell, C. Giraud-Carrier, and T. Martinez. Recommending learning algorithms and their associated hyperparameters. *International Workshop on Meta-learning and Algorithm Selection MetaSel*, 2014.

KA. Smith-Miles. Cross-disciplinary perspectives on metalearning for algorithm selection. *ACM Computing Surveys 41(6): 1-25.*, 2008.

J. Snoek, H. Larochelle, and R. Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems.*, 2012.

S. Spangler and J. Kreulen. *Mining the Talk: Unlocking the Business Value in Unstructured Information.* IBM Press, 2007.

S. Spangler and J. Kreulen. *Mining the Talk: Unlocking the Business Value in Unstructured Information.* IBM Press, 2008.

K. Srinagesh. *The principles of experimental research.* Butterworth-Heinemann, 2006.

V. Strijov and G. Weber. Nonlinear regression model generation using hyperparameter optimization. *Computers and Mathematics with Applications 60.4: 981-988.*, 2010.

G. Taguchi and S. Konishi. *Orthogonal Arrays and Linear Graphs.* Dearborn, MI: ASI Press, 1987.

G. Teng and M. Ho. Failure mode and effects analysis: an integrated approach for product design and process control. *International Journal of Quality and Reliability Management*, 1996.

C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. *In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 847-855)*, 2013.

G. Tononi. An information integration theory of consciousness. *BMC neuroscience 5.1 42.*, 2004.

K. Ulrich and S. Eppinger. *Product Design and Development.* Prentice Hall, 1995.

V. Vapnik and A Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications 16.2*, 1971.

D Ver Planck and B. Teare. *Engineering Analysis; An Introduction to the Professional Method.* Wiley, 1954.

P. Winston. *Artificial Intellignce.* Addison Wesley, 1992.

I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2005.