A Value Based Approach to the Development of Knowledge Engineering Systems that Integrates Product Design and Software Engineering: Methodology and Application

November 7, 2011

T. Munger¹, S. Desa¹

¹ Jack Baskin School of Engineering, University of California Santa Cruz

Abstract

There is an increasing trend across enterprises, technology or otherwise, to develop Knowledge Engineering systems to support core business activities, e.g. product development, customer support, and marketing. Frequently the deployment of these systems does not yield a valuable return on investment, either because they are not very useful to the users or the utility that they do deliver is not proportional to the amount of resources invested by the organization.

For the rapid and effective development of high-value Knowledge Engineering System it is critical to take an integrated approach value-focused approach to the design, development, and delivery of the system. To this end we propose an integrated framework for the development of high-value Knowledge Engineering software systems. Our novel contribution is the integration of Product Design methods such as the House of Quality, Function Structure analysis, Morphological Matrices, and Utility Functions to address value considerations throughout the
design and development process. Depending on the complexity of the problem and the time-
constraints the formal methods can often be simplified to increase cost-effectiveness.

The proposed framework has been used to develop a simple but non-trivial Knowledge
Engineering system for working with service request (support case) data in the computer-
networking domain. The developed system improved productivity when working with service
requests by over 30% compared to the previously used tools. Furthermore the structured
approach enabled by the framework allowed the system to be developed both quickly and at low
cost to the organization.

1. Introduction

Nearly all enterprises, technology and otherwise, are routinely collecting data and information
and attempting to transform it into knowledge that can be used to influence core business
activities, e.g. product development, customer support, and marketing. Knowledge Engineering
(KE) refers the methods and techniques for the extraction and retrieval of knowledge from data
and information. Two important areas of Knowledge Engineering are data mining, which
involves the extraction of knowledge from existing data and information, and information
retrieval, which is related to the problem of rapidly finding relevant knowledge. Because massive
quantities (terabytes to petabytes) of data and information are being collected, enterprises are
increasingly looking to automate this process through the creation of software based Knowledge
Engineering systems.

The overall objective when developing these Knowledge Engineering systems is to create a
system that is valuable to both the users and the organization. User value corresponds to the
systems quality or usefulness, while organizational value corresponds to the systems overall
costs (time and money). Many Knowledge Engineering systems are developed with insufficient attention to these two areas (quality and cost), resulting in systems that are low value to users because they are overcomplicated and difficult to use. Likewise when Knowledge Engineering systems do provide useful results, it is often at a significant cost (time and money) to the organization that outweighs the generated benefits. In order to concretely illustrate these issues, consider the following Knowledge Engineering problem in the computer-networking domain.

The development of automated diagnostic routines for next-generation smart network products and services is heavily dependent on knowledge contained within customer support cases or service requests (Figure 1). The network engineers responsible for developing the automated diagnostic routines read through large number (100s) of resolved service requests on a daily basis to understand: frequent problems experienced by customers and the steps taken to resolve these problems. This information is then synthesized into problem-solution pairs, which are used to develop automated diagnostic routines.

![Figure 1: Development of Automated Diagnostics for Smart Network Products and Services](image)

The productivity of the network engineers is severely impacted by the following two issues associated with locating and extracting problem-solution pairs from service requests.

- **Retrieval**: A search engine is used to locate service requests based on keyword matching within the service request text. However the relevance of a service request depends on a number of things (attachments, fields, etc.) not addressed in keyword matching. Consequently each search result must be manually evaluated for relevance, making it time consuming to locate service requests relevant to a specific problem.
• **Content Filtering:** In order to evaluate service requests for relevance and extract problem-solution pairs engineers have to currently read through many pages (20-50) of irrelevant email threads, poorly formatted text, and duplicated content to find the problem being experienced by the customer and the steps taken to resolve the problem.

Developing a Knowledge Engineering system to address the retrieval and content filtering issues discussed above requires methods and techniques from three domains (Figure 2). The domain of Knowledge Engineering is necessary to address the knowledge aspects of the system, in particular, modeling the intended users (customers) work process to ensure that the system provides useful functionality. The domain of Product Design is necessary to address the usability and cost aspects of the, in particular designing a high-quality product that satisfy the user needs and also minimize development costs (time and money). Finally, the Software Engineering domain is needed to develop the product into a robust and reliable software based system that can be deployed to the users.

![Figure 2: Three Domains Involved in Developing High-Quality Knowledge Engineering systems](image-url)
In order to successfully develop a high-value Knowledge Engineering system it is not sufficient to use a “throw it over the wall” approach where the each domain is addressed independent of the other domains. Delivering value to both the users and the organization involves making trade-offs between the system’s quality and cost and therefore requires close coordination and cooperation between the three domains. In this paper we will show how the methods and techniques from each of these domains can be integrated and used to develop Knowledge Engineering systems. Our novel contribution is the integration of formal Product Design methods to explicitly address important issues that come up during the development of Knowledge Engineering systems—such as relating the user to the system, thinking about the system functionally, exploring different function realizations, and selecting the best realization based on the user needs and cost objectives—that are not generally given enough attention in conventional development methods. Addressing these issues in a Product Design environment, as opposed to a Software Engineering environment, allows us to converge on a high-quality design sooner, minimizing the number of costly software revisions necessary to produce a Knowledge Engineering system that sufficiently satisfies the users’ needs. We have demonstrated these results within the context of the network-engineering problem described earlier.

The paper is structured as follows: In Section 2 we survey the relevant methods and techniques from each domain and outline the issues involved in integrating these methods and techniques into a framework for developing Knowledge Engineering systems. Section 3 surveys existing literature that addresses integration the domains and evaluates their applicability towards our objective. Section 4 describes the three phases of our integrated framework for developing high-value Knowledge Engineering systems. Section 5 lists possible simplifications to the
framework in order to address situations when the Knowledge Engineering problem is of low complexity or there are time constraints that prohibit the use of the full framework. Section 6 discusses the results of applying the framework to simple but non-trivial problem in the computer-networking domain. Our conclusion and path for future work is presented in Section 7.

2. Problem Description

In this section we outline the issues involved combining methods and techniques from Knowledge Engineering, Product Design, and Software Engineering in a new value centered methodology for developing software based Knowledge Engineering systems. We begin by providing a brief overview of the kernel work in each domain that will form core of our integrated framework. We then describe the integration issues that must be addressed before these methods and techniques can be used as a cohesive framework.

In the domain of Knowledge Engineering we will focus on work process modeling [8]. Work process modeling is used to capture the existing work process that the systems intended users (customers) use to solve problems. Once this process is well understood then we can identify areas that are good candidates for optimization. There are a number of other areas in Knowledge Engineering, such as Data Mining [9][10][11] and Information Retrieval [13], which are important to the development of Knowledge Engineering systems but outside the scope of this paper.

In the domain of Product Design, we draw on four well-known techniques used in conceptual design. The House of Quality [14] is used to translate the user (customer) needs into measurable engineering characteristics that can be used by the development team to design the product. The Function Structure [15] captures the functions that the product will need to perform in order to
satisfy the user needs. The Morphological Matrix [15] technique is used to explore multiple alternative realizations for the products functions. Finally, the Utility Function [16] technique provides an objective way of selecting the design that best satisfies the customer needs and other objectives (costs, time, etc.) for the product.

In the Software Engineering domain we will focus on three Unified Modeling Language (UML) diagrams commonly used in object oriented design. Use case diagrams [20] are used to understand how the users will interact with the system. Component diagrams [22] are used to define the high-level software modules of the system and their respective interfaces. Class diagrams [22] are used to decompose each module into a set of objects that represent the data and functions. There are a number of other useful UML models, such as the Activity diagram [22], however they are not as critical to the development of Knowledge Engineering systems and are also outside the scope of this paper.

There are two central issues that must be resolved before these methods and techniques can be used together as an integrated framework. First we must address the intra-domain integration, or the unification of the individual methods and techniques within each domain. Although the methods and techniques within each domain are generally compatible with each other, their interfaces are not always well specified. For example, although Product Design literature[16][17][18] discuss the techniques such as the House of Quality and Function Structure in great detail they do not provide explicit processes for moving between the these two techniques. Therefore we will need to develop such processes for interfacing between the methods and techniques used in each domain. Second, we must address the inter-domain integration, or combining the methods and techniques across the three domains into a complete framework (methodology) that can be used within the context of developing Knowledge
Engineering systems. Currently the domains are, for the most part addressed independently. Therefore we will need to create processes for using the results (output) of each domain to drive the work in the proceeding domains.

3. Related Work

In this section we review existing literature that addresses integration between the Knowledge Engineering, Product Design, and Software Engineering domains and evaluate their applicability/utility to our objective of developing high-value Knowledge Engineering systems.

To our knowledge, there is no existing work that fully integrates Knowledge Engineering, Product Design, and Software Engineering within the context of developing Knowledge Engineering systems. Existing work that partially integrates these domains falls into two categories: work that address the integration of Knowledge and Software Engineering, and work that addresses the integration Software Engineering and Product Design. An overview of these approaches with respect to the three domains is provided in Table 1 below.
Table 1: Comparison of Integrative Approaches

<table>
<thead>
<tr>
<th>Approach/ Criteria</th>
<th>Knowledge Engineering</th>
<th>Product Design</th>
<th>Software Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common KADS</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>(Schrieber, et. al)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protégé</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>(Gennari, et. al)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VBSE</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>(Bohem, et. al.)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The modeling approach focuses creating Knowledge Engineering models that can be used to drive the Software Engineering process. Common Knowledge Application and Discovery (KADS) framework [2], is a popular modeling framework which has been used in a wide range of applications ranging from smart spatial sensors [3] to medical diagnosis [4]. Common KADS provides an interconnected set of tools for modeling with three tasks: identifying valuable organizational knowledge, capturing and modeling that knowledge, and finally incorporating the knowledge into Knowledge Engineering systems. The Common KADS framework is useful for identifying organizational opportunities for Knowledge Engineering systems and structuring knowledge intensive work processes. However the Common KADS provides limited guidance on how these models can be used to drive the design and implementation of Knowledge Engineering systems.
The component approach is based around the idea that there are certain components, such as the knowledge base, user interface, etc., that are common across most Knowledge Engineering systems. The component based approach therefore focuses on creating software implementations of these components that can be reused across a number of Knowledge Engineering systems. The Protégé framework [1] is a popular component based framework for building knowledge bases. Protégé provides tools for allowing domain experts to capture their knowledge through the creation of domain ontologies, which can then be used to construct knowledge bases. Although component based frameworks, such as the Protégé, can be used to accelerate the development of Knowledge Engineering system, they don’t address the fundamental problem of designing a Knowledge Engineering system that provides value to both the users and the organization.

Work in the area of Value Based Software Engineering (VBSE) [5][6][7] address how value considerations can be integrated into Software Engineering theory and practice. VBSE is based around 7 key principles such as Business Case Analysis, Concurrent Engineering, and Agile development. While these principles are useful when developing conventional software system, they are of less utility when developing Knowledge Engineering systems because most of key decisions affecting the system’s value are performed before the Software Engineering phase. Therefore attempting to address value considerations during the Software Engineering will have a marginal impact on the overall value of the system. Other approaches to integrating Product Design and Software Engineering generally focus on the on the House of Quality technique. For example, the House of Quality has been applied to task of prioritizing user requirements [21]. However these applications are limited in scope, they only address specific aspects of the Software Engineering process, and are rarely used in practice.
4. Integrated Framework for Developing High-Value Knowledge Engineering Systems

In this section we describe our integrated framework for developing high-value Knowledge Engineering systems. The framework consists of three phases (Figure 3). In Phase I we address the Knowledge aspects of the system and derive the Knowledge Engineering objectives for the system. In Phase II these objectives are used to design a Knowledge Engineering product that addresses users’ needs. In Phase III the product is implemented as software based system. In order to illustrate our ideas and techniques we will use the computer networking example from the Section 1 as a running example to illustrate the frameworks practical application.

The emphasis and primary contribution of our framework is the integration of the Product Design methods in Phase II with the Software Engineering in Phase III. Phase I is a simplified version of a Common KADS Knowledge Engineering, while Phase III is a simplified version of a conventional Unified Modeling Language (UML) process.

Phase I: Knowledge Engineering

For the purposes of this paper we will focus on two essential parts of the Knowledge Engineering process: modeling the users’ work process and defining a set of objectives from that work process. The resulting objectives will then be used to drive the design of the Knowledge Engineering product in Phase II.
**Work Process Model**

The first task in the Knowledge Engineering phase of our framework is to develop a model of the users’ current work process [8]. Using this model we can then identify areas of the work process that would benefit from Knowledge Engineering and/or automation. The process for developing the work process model is as follows:

1. **Use interviews and questionnaire to gather information about the user’s current work process.** Start by capturing high-level information such as what the users are trying to accomplish (the overall output of the work process). Next identify the tasks in the work process and the inputs and outputs to each task. Finally determine characteristics of each task such as relative difficulty, duration, frequency, etc.

The engineers currently locate service requests using an enterprise search engine that indexes the service request database (Figure 4). Engineers provide a set of keywords and the search engine will return a set of service requests that contain the keywords. To assess the “real” relevance of each search result, the engineers need to open and review (speed-read) the service request. The large number of results returned by search engine (typically queries return 100-10,000 service requests) makes even a brief review of each service request an extremely laborious task. Once a service request is determined to be relevant then the service request in question has to be read more thoroughly in order to extract the problem-solution pair. The excessive length (20-50 pages) and free form structure of the service requests make careful reading a difficult and tedious task.

![Figure 4: Engineer Work Process](image-url)
2. Document the responses in the form of a work process.

_Engineers Work Process_

1. Run a keyword search using service request database search engine to locate Service requests
2. Locate relevant Service requests in the search results
   2.1. Open the service request using the service request Viewer
   2.2. Examine service request fields (technology, sub-technology)
   2.3. Locate and read the problem description
   2.4. Locate and read resolution summary
   2.5. Locate and read first and last correspondence
3. Extract problem solution pairings from relevant Service requests
   3.1. Re-examine the problem description, resolution summary, and first and last correspondence in more detail
   3.2. Read case notes if necessary
   3.3. Download and read attachments if necessary
4. Use the problem-solution pairs to create smart network products

_**Knowledge Engineering Objectives**_

Once we have developed a model of the user work process the next step is to determine the Knowledge Engineering objective, e.g. what are we trying to accomplish with the Knowledge Engineering system. The process for determining the Knowledge Engineering objective is as follows:

1. **Identify tasks (steps) in the work process that will benefit the most from automation, e.g. manual repetitive processes.**

   A significant amount of the engineers’ time was spent doing repetitive tasks related to locating relevant service requests and extracting problem-solution pairs. For example, the engineers have to manually examine the service request fields in order to determine if the service request is relevant.

2. **Based on the identified tasks define the overall objective for the Knowledge Engineering system.**

   It was determined the overall objective of the Knowledge Engineering system would be to “facilitate searching for relevant service requests and extracting problem-solution pairs”. A high level overview of the desired system is shown in Figure 5.
Phase II: Product Design

In this phase we design a Knowledge Engineering product to satisfy the user needs and address other development objectives such as time and cost.

User Needs
User needs [14][18] capture what the users (customers) desire from the product. We organize the user needs for the Knowledge Engineering product into three high level categories: functional, usability, and integration. Functional needs describe the functions that product must perform. Usability needs describe characteristics that the product must embody such as easy to use. Integration needs describe how the product must be integrated into the users’ existing work process. We will focus on explicitly identifying the functional and usability needs. The integration needs will be brought out implicitly through the functional needs. The resulting set of user needs is then related to the technical metrics (engineering characteristics) for the product in the House of Quality. The process for determining the user needs for the product is as follows:
1. Define the functional needs for the product based on the overall objective(s) identified during the Knowledge Engineering phase. Each functional need should describe a function that the product provides to the users.

In the Knowledge Engineering phase we determined the overall objective of the Knowledge Engineering system would be to “facilitate searching for relevant service requests and extracting problem-solution pairs”. This objective translated to the two primary functional requirements: “locating relevant service requests” and “extracting problem-solution pairs”. The secondary functional requirements were derived by examining the individual tasks that the engineers performed in each step. For example, in order to locate relevant service requests the engineers manually examined the fields (technology, sub-technology) of each service request. The engineers’ wanted to be able to specify these field values as part of the search criteria, so that they didn’t have to manually filter search results. This user need was captured in the secondary functional requirements “be able to do very targeted searches”.

2. Interview users to identify usability needs that are important to the users. Each need should describe a product characteristic desired by the users.

When the engineers were interviewed about what product characteristics were most important to them, they expressed that it was critical that the product introduce a minimum amount of overhead into their current work process. In order to capture this need we used two high level usability requirements: “easy to use” and “high performance”.

3. Work with the users (customers) to establish the relative importance of each need using a convenient scale (e.g. 1-10).

Each user need was ranked according to its respective impact on the work process (Table 1). For example, the user need “be able to do very targeted searches” was ranked high (8/10) because it would allow the engineers to reduce the number Service requests they had to read and therefore significantly improve their productivity. On the other hand the user need “high performance” was ranked relatively low (3/10) because performance was not extremely important as long as it didn’t add significant overhead (more than a couple of minutes) to the work process.
### Table 2: Prioritized list of User Needs for the Knowledge Engineering product

<table>
<thead>
<tr>
<th>User Need</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Be able to do broad searches</td>
<td>5</td>
</tr>
<tr>
<td>Be able to do very targeted searches</td>
<td>8</td>
</tr>
<tr>
<td>Be able to read service requests easier</td>
<td>7</td>
</tr>
<tr>
<td>Be able to quickly assess the relevance of a service request</td>
<td>8</td>
</tr>
<tr>
<td>Be able to quickly extract problem-solution pairs</td>
<td>7</td>
</tr>
<tr>
<td>Seamless access to service request attachments</td>
<td>5</td>
</tr>
<tr>
<td>Easy to use</td>
<td>5</td>
</tr>
<tr>
<td>High performance</td>
<td>3</td>
</tr>
</tbody>
</table>
4. Organize the user needs (functional and usability) into a hierarchy of primary needs, secondary needs, and tertiary needs.

Figure 6: User Needs Hierarchy for the Knowledge Engineering product

**Technical Metrics**

In order to design a high-quality product it is necessary for the design and development team to have a clear idea of what is expected of the product. User needs such as “be able to read service requests easier” and “high performance” can be relatively subjective, different people might have different interpretations of “high performance”, which makes them difficult to use when designing and developing the product. Technical metrics [14][18] are precise, quantitative metrics that allow the design and development team to objectively measure how well the product satisfies the user needs. Each technical metric consists of a measurement as well as target value. For example, the metric “average time to assess relevance of an service request” has a target value of 5 minutes. In this section we provide the process for developing the technical metrics. The relationships between the user needs and technical metrics are then explicitly captured in the House of Quality, which is used extensively throughout the Product Design and Software
Engineering phases. The process for developing the technical metrics for the product is as follows:

1. **Make a list of technical metrics for measuring each functional and usability needs.** For the functional need, consider the steps in the users’ work process and try to develop a set of metrics that capture the factors that influence the work process. For the usability requirements consider how the user will interact and use the system, and try to capture the factors that will influence these interactions.

   In order to measure the functional need “be able to read Service requests easier” we defined two quantitative metrics: “number of pages read to assess relevance” and “average time to assess relevance”. Likewise, in order to measure the usability need “easy to use” we defined the metric “average number of icons in the Graphical User Interface”.

2. **Determine a target value for each technical metric.** The target values for the functional needs should represent a significant improvement to the current work process. The target values for the usability needs should be based on user input as well as dissecting any currently used software tools or systems.

   The target values for the metrics related to the functional needs were determined by examining the users current work process (Table 2). The target values for the technical metrics related to usability need, e.g. “easy to use” and “high performance”, were determined by dissecting software tools currently used by the engineers.

<table>
<thead>
<tr>
<th>Technical Metric</th>
<th>Target Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of pages read to assess relevance</td>
<td>10 pages</td>
</tr>
<tr>
<td>Average time to assess relevance</td>
<td>5 minutes</td>
</tr>
<tr>
<td>Average time to extract problem solution pair</td>
<td>20 minutes</td>
</tr>
<tr>
<td>Average steps to complete typical task</td>
<td>3 steps</td>
</tr>
<tr>
<td>Number of icons in the user interface</td>
<td>10 icons</td>
</tr>
<tr>
<td>Average time to run broad search</td>
<td>1 minute</td>
</tr>
<tr>
<td>Average time to run targeted search</td>
<td>2 minutes</td>
</tr>
</tbody>
</table>
**House of Quality**

In order to ensure that the product and its software system implementation sufficiently address the users’ needs it is necessary that the relationship between user needs and technical metrics (engineering characteristics) be clearly understood. The House of Quality (HOQ) [14] relates the user needs for the product to technical metrics in a simple easy to use diagram. The HOQ is used extensively through the Product Design and Software Engineering phases in order to ensure that the user needs for the product are being satisfied. The process for creating the product’s HOQ is as follows:

1. **Identify relationships between the user needs and technical metrics using a convenient scale such as strong, moderate, weak, and no relationship.** A strong relationship indicates that increasing or decreasing the technical metric will affect the user need. No relationship means that the user need and technical metric are independent of each other—changing the technical metric will have no effect on the user need. The result is the relationship matrix.

   The user need “be able to read service requests easier” is strongly related to the technical metric “number of pages read to assess relevance”. It is also moderately related to the “average time to assess relevance”.

2. **Correlate the technical metrics to each other using a convenient scale (e.g. positive or negative correlation).** A positive correlation means that the technical metrics increase together, while a negative correlation means that the technical metrics decrease together. The result is the correlation matrix.

   The technical metric “number of pages read to assess relevance” is positively correlated to the “average time to assess relevance”.

3. **Combine the user needs, technical metrics, relationship matrix, correlation matrix into a House of Quality diagram according to Figure 7 below.** The relationship matrix occupies the center of the House of Quality, with the user needs are to the left and the technical metrics are above. The correlation matrix is positioned on top of the technical metrics. Finally the target values for the technical metrics are positioned below the relationship matrix.
Figure 7: House of Quality for the Knowledge Engineering product

Function Structure

The user needs captured in the House of Quality are generally not detailed or comprehensive enough to design a product from. The Function Structure [15] provides a detailed solution-neutral representation of the product functions and sub-functions necessary to implement the functional requirements specified in the House of Quality. The Function Structure ensures that the design is functionally complete and enables (facilitates) the generation of a several feasible alternative product concept realizations. The process for creating the Function Structure for the product is as follows.

1. Define the main function for the product based on objective(s) identified in the Knowledge Engineering phase. Determine the information inputs and outputs to the primary function based on the work process steps to be automated. This is the first level of the Function Structure.

The Knowledge Engineering objective “facilitate searching for relevant service requests and extracting problem-solution pairs” was used as the main function for the product.
2. Decompose the primary function into a set of sub-functions that correspond to the primary functional needs in the HOQ. There should be at least one sub-function for each need. This is the second level of the Function Structure.

The two primary functional needs, “locate relevant service requests” and “extract problem-solution pairs”, are captured by the second level sub-function “locate relevant service requests” and “extract problem-solution pairs” in the Function Structure.

3. Continue this decomposition process—i.e. the third level sub-functions should correspond to the secondary functional need in the HOQ—until each sub-function is simple enough to realize.

The Function Structure for the Knowledge Engineering product has four levels (Figure 8). The first level consists of the main function “facilitate searching for relevant service requests and extracting problem-solution pairs”. The second level sub-function “locate relevant service requests” can be broken up into the third level sub-functions “locate candidate service requests” and “view candidate service requests”. The third level sub-function “locate candidate service requests” can be further broken up into the fourth level sub-functions “keyword search service request content”, “search service request fields”, and “search service request attachments”.

**Figure 8: Function Structure for the Knowledge Engineering product**

![Function Structure Diagram]

**Morphological Matrix**

Once the product’s sub-functions have been identified, the next step is to determine how these sub-functions will be implemented. The Morphological Matrix [15] provides a systematic method to capture the space of feasible solution principles (realizations) for the product sub-functions specified in the Function Structure. The structured format of the Morphological Matrix enables the rapid generation of product concepts for further investigation and will frequently result in product concepts that might not of have been considered in less systematic methods. The process for creating a Morphological Matrix for the product is as follows:

1. **Generate several (2-5) alternative solution-principles for realizing each leaf sub-function (terminal node with no children) in the Function Structure.**

   One important leaf sub-function in the product’s Function Structure is “remove duplicate content”. There are several possible approaches to realizing this sub-function. A simple approach would be to use an exact-match hash function to detect duplicate paragraphs. A slightly more flexible approach would be to use a “fuzzy” hash function based off the similarity of the paragraphs. An even more sophisticated approach would be to train a classifier to detect characteristics of duplicate paragraphs. For example, textual features such as ‘>’ are often indicative of a repeated email thread.
2. Organize the sub-functions and solution-principles in a matrix—where the sub-functions are the matrix rows and the solution-principles are the matrix columns.

Figure 9: Morphological Matrix for the Knowledge Engineering product

<table>
<thead>
<tr>
<th>Sub-Function</th>
<th>Solution Principle 1</th>
<th>Solution Principle 2</th>
<th>Solution Principle 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword search SR content</td>
<td>Database access and cosine similarity based keyword search</td>
<td>Database access and regular expression keyword search</td>
<td>Leverage search engine results</td>
</tr>
<tr>
<td>Search SR fields</td>
<td>Augment search engine fields</td>
<td>Keyword search fields</td>
<td>Leverage search engine fields</td>
</tr>
<tr>
<td>Search SR attachments</td>
<td>Keyword and search by attachment type</td>
<td>Search by attachment type</td>
<td>Keyword search</td>
</tr>
<tr>
<td>Retrieve SR content</td>
<td>Database access</td>
<td>Search engine XML interface</td>
<td></td>
</tr>
<tr>
<td>Remove repeated content</td>
<td>Use a classifier to detect duplicate information</td>
<td>Fuzzy hash based deduplication</td>
<td>Strict hash based deduplication</td>
</tr>
<tr>
<td>Summarize SR contents</td>
<td>Natural language processing</td>
<td>Extract problem description and resolution summary, Remove stop words and text noise</td>
<td>Extract problem description and resolution summary.</td>
</tr>
<tr>
<td>Access to attachments</td>
<td>Attachments displayed in user web browser with attachment type and keyword highlighting</td>
<td>Attachments displayed in user browser</td>
<td>User downloads attachments</td>
</tr>
<tr>
<td>Access to SR notes</td>
<td>Keyword highlighting with collapsible panes for each note type</td>
<td>Organized by note type</td>
<td>Plain text</td>
</tr>
</tbody>
</table>

3. Generate 2-3 alternative product concepts by suitably combining the solution-principles in the Morphological Matrix. Suitably refers to ensuring that the
selected solution-principles are compatible with each other and have a consistent focus (high quality, low cost, etc.).

The Morphological Matrix (Figure 10) was used to generate three alternative product concepts for the Knowledge Engineering system. Product concept 1 represents a high-quality high-cost product that uses complex Information Retrieval techniques such as natural language processing built on top of a custom built software infrastructure. Product concept 2 represents a low-quality low-cost product that uses very simple Information Retrieval technical such as regular expression matching. Product concept 3 represents a hybrid approach that builds on the existing functionality provided by existing tools (the enterprise search engine) with relatively simple Information Retrieval techniques.
Figure 10: Three Alternative Product Concepts for the Knowledge Engineering product

<table>
<thead>
<tr>
<th>Sub-Function</th>
<th>Solution Principle 1</th>
<th>Solution Principle 2</th>
<th>Solution Principle 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword search SR content</td>
<td>Database access and cosine similarity based keyword search</td>
<td>Database access and regular expression keyword search</td>
<td>Leverage search engine results</td>
</tr>
<tr>
<td>Search SR fields</td>
<td>Augment search engine fields</td>
<td>Keyword search fields</td>
<td>Leverage search engine fields</td>
</tr>
<tr>
<td>Search SR attachments</td>
<td>Keyword search by attachment type</td>
<td>Search by attachment type</td>
<td>Keyword search</td>
</tr>
<tr>
<td>Retrieve SR content</td>
<td>Database access</td>
<td>Search engine XML interface</td>
<td></td>
</tr>
<tr>
<td>Remove repeated content</td>
<td>Use a classifier to detect duplicate information</td>
<td>Fuzzy hash based deduplication</td>
<td>Strict hash based deduplication</td>
</tr>
<tr>
<td>Summarize SR contents</td>
<td>Natural language processing</td>
<td>Extract problem description and resolution summary, Remove stop words and text noise</td>
<td>Extract problem description and resolution summary</td>
</tr>
<tr>
<td>Access to attachments</td>
<td>Attachments displayed in user web browser with attachment type and keyword highlighting</td>
<td>Attachments displayed in user browser</td>
<td>User downloads attachments</td>
</tr>
<tr>
<td>Access to SR notes</td>
<td>Keyword highlighting with collapsible panes for each note type</td>
<td>Organized by note type</td>
<td>Plain text</td>
</tr>
</tbody>
</table>
Selection Criteria

By combining solution-principles in Morphological Matrix we have created several product concepts that could be developed into the Knowledge Engineering system. The next step is to select a product concept that provides the highest value with respect to users and organization. The Selection Criteria [16] capture these objectives (factors) that influence the product value in a structured format so that they can be used to evaluate the generated product concepts. The Selection Criteria are used by the Utility Function to calculate a cumulative value for each product concept. The process for generating the selection criteria for the product selection is as follows:

1. Generate a hierarchy (tree) of objectives for the product. Each level of the hierarchy describes the product objectives at an increasing level of detail. At the first level of the hierarchy is the overall objective, e.g. develop a high-value Knowledge Engineering product. At the second level there are typically three objectives: quality, complexity (development time), and cost. The quality objectives for the following levels should be derived based on the user needs in the House of Quality. The complexity objectives from the sources of product complexity (e.g. knowledge engineering, software infrastructure, etc.). Finally cost objectives should capture any costs associated with the solution-principles (e.g. buying commercial software packages).

Figure 21: Utility Function Objective Hierarchy for the Knowledge Engineering product
2. Assign each objective in the hierarchy a relative weight $w^r$ according to how important it is relative to the other objectives. The relative weights should be assigned so that relative weights of every objective’s immediate descendants sum to 1. The relative weights of the quality objectives are determined by the relative importance of each user need. The weights for the other objectives (complexity, cost, etc.) should be assigned while working in collaboration with users, project stakeholders, and the software developers who will be implementing the product as a software system.

Figure 32: Utility Function Relative Weights for the Knowledge Engineering product

3. Determine the absolute weight of each objective by multiplying the objective’s relative weight $w^r$ by the absolute weight $w^a$ of its parent. The absolute weights for each level must sum to 1.

$$w^{a}_{11} = w^{r}_{11} \times w^{a}_{1}$$

Figure 13: Utility Function Absolute Weights for the Knowledge Engineering product
Utility Function

The next step is to apply the Selection Criteria in order to select the “best” product concept. The Utility Function [16][17] is a mathematical technique for calculating overall utility or usefulness for each generated product concept with respect to development objectives (quality, time, cost, etc.) specified in the Selection Criteria. The product concepts can then be quantitatively compared with respect to their overall utility in order to select the “best” product concept. The selected product concept is then developed into a software system during the Software Engineering phase. The process for creating a Utility Function for the product is as follows:

1. Select the leaf objectives absolute weights from the Selection Criteria tree and arrange them into a table with the product concepts. The objectives go in the table rows, and the product concepts go in the table columns.

![Utility Function for the Knowledge Engineering Product](image)

<table>
<thead>
<tr>
<th>Objectives</th>
<th>Weights</th>
<th>Product Concept 1</th>
<th>Product Concept 2</th>
<th>Product Concept 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide broad search capabilities</td>
<td></td>
<td>Score</td>
<td>Utility</td>
<td>Score</td>
</tr>
<tr>
<td>Provide targeted search capabilities</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Be able to quickly assess the relevance of the service request</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy access to attachments</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to find problem in service request</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to find resolution in service request</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity of Knowledge Engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity Software Infrastructure</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cumulative Utility</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Assign each product concept $P^i$ a score $s^i_j$ for each objective $o_j$, using a convenient scale (e.g. 1-10), based on how well the $P^i$ satisfies $o_j$.

Figure 5: Product Concept Scores for the “Provide Broad Search Capabilities” Objective

<table>
<thead>
<tr>
<th>Objectives</th>
<th>Weights $(w^a_j)$</th>
<th>Product Concept 1</th>
<th>Product Concept 2</th>
<th>Product Concept 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide broad search caps.</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

3. Calculate the utility scores for each design concept by multiplying the selection criteria scores by their corresponding weights. Compute a cumulative utility $CU^i$ for each product concept $P^i$ by summing the utility scores.

$$CU^i = \sum_{j=1}^{\text{number of objectives}} s^i_j w^a_j$$
Figure 6: Cumulative Utility Scores for the Product Concepts

<table>
<thead>
<tr>
<th>Objectives</th>
<th>Weights ((w_j ))</th>
<th>Product Concept 1</th>
<th>Product Concept 2</th>
<th>Product Concept 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Score</td>
<td>Utility</td>
<td>Score</td>
<td>Utility</td>
</tr>
<tr>
<td>Provide broad search capabilities</td>
<td>0.042</td>
<td>8</td>
<td>0.336</td>
<td>4</td>
</tr>
<tr>
<td>Provide targeted search capabilities</td>
<td>0.126</td>
<td>6</td>
<td>0.756</td>
<td>3</td>
</tr>
<tr>
<td>Be able to quickly assess the relevance of the service request</td>
<td>0.21</td>
<td>7</td>
<td>1.47</td>
<td>4</td>
</tr>
<tr>
<td>Easy access to attachments</td>
<td>0.044</td>
<td>4</td>
<td>0.176</td>
<td>2</td>
</tr>
<tr>
<td>Easy to find problem in service request</td>
<td>0.072</td>
<td>7</td>
<td>0.504</td>
<td>3</td>
</tr>
<tr>
<td>Easy to find resolution in service request</td>
<td>0.072</td>
<td>7</td>
<td>0.504</td>
<td>3</td>
</tr>
<tr>
<td>Complexity of Knowledge Engineering</td>
<td>0.28</td>
<td>3</td>
<td>0.84</td>
<td>8</td>
</tr>
<tr>
<td>Complexity Software Infrastructure</td>
<td>0.12</td>
<td>3</td>
<td>0.36</td>
<td>4</td>
</tr>
<tr>
<td>Cumulative Utility (CU^I)</td>
<td>4.946</td>
<td>4.626</td>
<td>5.984</td>
<td></td>
</tr>
</tbody>
</table>

**Product Concept**

The product concept with the highest cumulative utility is then selected for implementation in the Software Engineering phase. Table 4, below, lists the sub-function and corresponding solution principles of the selected product concept.
Table 4: Selected Product Concept for the Knowledge Engineering product

<table>
<thead>
<tr>
<th>Sub-Function</th>
<th>Solution Principle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search service request content</td>
<td>Leverage search engine results.</td>
</tr>
<tr>
<td>Search service request fields</td>
<td>Augment search engine fields.</td>
</tr>
<tr>
<td>Search service request attachments</td>
<td>Keyword search by attachment type.</td>
</tr>
<tr>
<td>Retrieve service request content</td>
<td>Search engine XML interface.</td>
</tr>
<tr>
<td>Remove repeated content</td>
<td>Fuzzy hash based deduplication.</td>
</tr>
<tr>
<td>Summarize service request contents</td>
<td>Extract problem description and resolution summary. Remove stop words and textual noise.</td>
</tr>
<tr>
<td>Access to attachments</td>
<td>Attachments displayed in user web browser with attachment type and keyword highlighting</td>
</tr>
<tr>
<td>Access to service request notes</td>
<td>Keyword highlighting with collapsible panes for each note type</td>
</tr>
</tbody>
</table>

Phase III: Software Engineering

During the Software Engineering phase the selected product concept is translated into a software specification (software design) and then implemented as a software system (software development). We are essentially following Software Engineering conventions with the exception that we are radically altering the drivers (inputs) that feed into the software design and development processes.

We have developed several processes for ensuring that the product concept is accurately translated to the Software Engineering. For software design we have defined a process for bridging between Product Design artifacts (Function Structure, Morphological Matrix, etc.) and
Unified Modeling Language (UML) methods for software design. For software development we have defined a process for incorporating the Product Design artifacts (House of Quality, Function Structure) into the software planning and testing processes in order to ensure the developed software sufficiently satisfies the user needs.

**Software Design**
In this phase the selected product concept is used to drive the software design of the Knowledge Engineering system.

**Use Case Diagram**
In order to successfully translate the product concept into a software based Knowledge Engineering (KE) system, it is first necessary to understand how the users will interact with the product. Use Case diagrams [19][20] are a standard software design technique for identifying interactions between the user and the software system. Each Use Case diagram consists of four elements: the actors (such as users and external systems) that interact with the KE system, the KE system itself, the interactions that the actors can have with the KE system, and the lines that represent the relationships between these elements. The Use Case diagrams are used to define the Graphical User Interface (GUI) in Component diagram. The process for creating the Use Case diagram for the KE system is as follows:

1. **Create a Use Case diagram for each second sub-function in the Function Structure.** The Use Case diagram should be based around a single goal that the user that the user is trying to accomplish by using that function.
   
The Use Case diagram “search for service request” corresponds to the second-level sub-function “locate relevant service requests” in the Function Structure.

2. **Add the interactions for achieving the goal based on the sub-functions below the high-level sub-function.** Each interaction should roughly correspond to sub-function in the Function Structure.
   
   There are three interactions that must be considered when the user is searching for Service: “enter search keywords”, “enter service request field values”, and “enter attachment keywords”. These interactions corresponded to the Function Structure
sub-functions “keyword search service request content”, “search service request fields”, and “search service request attachments”.

3. **Identify the relationships between the actors (users and external systems) and the interactions.** Functional dependencies, where one interaction requires the other interaction, are labeled with the <<uses>> relationship. Variations, where one interaction is a special case of another interaction, are labeled with the <<extends>> relationship.

*Figure 7: Use Case Diagram for the “Search for Service Request” Use Case*

![Use Case Diagram](image)

**Component Diagram**

The Component diagram [22] defines the high-level architecture for the software system in terms of the software components (modules) and their relationships. We later use the Component diagram to drive the detailed software design, and create a software development plan. The process for creating the component diagram is as follows:

1. **Group the product concept functionality (sub-function, solution-principles pairs) into sets of related functionality using the hierarchy defined in the Function Structure.**
Table 5: Product Concept Functionality Groups for the Knowledge Engineering system

<table>
<thead>
<tr>
<th>Group</th>
<th>Sub-Function</th>
<th>Solution Principle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search Functionality</td>
<td>Search service request content</td>
<td>Leverage search engine results.</td>
</tr>
<tr>
<td></td>
<td>Search service request fields</td>
<td>Augment search engine fields.</td>
</tr>
<tr>
<td></td>
<td>Search service request attachments</td>
<td>Keyword search by attachment type.</td>
</tr>
<tr>
<td>Content Functionality</td>
<td>Remove repeated content</td>
<td>Fuzzy hash based deduplication.</td>
</tr>
<tr>
<td></td>
<td>Summarize service request contents</td>
<td>Extract problem description and resolution summary. Remove stop words and textual noise.</td>
</tr>
<tr>
<td>External Interface Functionality</td>
<td>Retrieve service request content</td>
<td>Search engine XML interface.</td>
</tr>
</tbody>
</table>

2. Define a functional component (module) for implementing each set of functionality. Identify the inputs and outputs to each software component using the information flows between the sub-functions (captured in the Function Structure).

The three translated to three functional software components: the search filter, the content filter, and the search engine interface. The search filter was defined to implement search functionality group, the content filter to implement content functionality group, and the search engine interface to implement the retrieval of service request content.

3. Define the necessary GUI components for facilitating user interaction with the system based on the Use Case diagrams. Typically there will be one GUI component for each Use Case. Identify the inputs and outputs to these GUI components based on the Use Case.

There were two GUI components (search and content) necessary for facilitating user interaction with the functional components. These GUI components were derived from the two Use Case diagrams (search for service requests, and read service requests).

4. Draw a system boundary around the functional and GUI components. Place the user to left of the system boundary and other supporting systems (databases, search engines, etc.) to the right. List the information flows between the user, the GUI components, and the functional components. The result is a Component diagram
Figure 8: Component Diagram for the Knowledge Engineering system

Class Diagram

The Component diagram does not provide enough detail to guide the software development. Each component will typically consist of a number of objects (classes) that are not captured in the Component diagram. The Class diagram [20][22] captures the details—methods and attributes—of these objects. The process for creating the class diagrams for each component in the component diagram is as follows:

1. **Define the necessary data structure classes to represent the component’s information inputs and outputs.**

   The Content Filter component operates on search results provided by the user and returns filtered service requests. This requires two data structures: SearchResult object to represent the search results and ServiceRequest object to represent the service requests.

2. **Define the functional classes necessary for transforming the input information into the output information.**

   The transformation of ServiceRequests into filtered ServiceRequests is handled by the Filter class. There are two different was implementations of the Filter class: the DeduplicationFilter for removing duplicate content and the SummarizationFilter for summarizing content.

3. **Define the necessary control classes for receiving inputs, triggering the necessary functions, and sending outputs.**

   The ContentFilter class receives SearchResults objects, retrieves the corresponding ServiceRequest objects, applies the DeduplicationFilter and SummarizationFilter, and returns the filtered ServiceRequest object.
4. Define the relationships (inheritance, aggregation, association) between classes. Use these relationships to organize the data structure classes, functional classes, and control classes. The result is the Class Diagram.

The ContentFilter class included two Filters for summarizing the ServiceRequest objects, a DeduplicationFilter and SummarizationFilter.

**Figure 99: Class Diagram for the Content Filter Component**

**Software Development**

During Software Development we translate the Software Design (Component and Class Diagrams) into executable code.

**Planning**

In order to effectively implement Knowledge Engineering system, the software architecture specified in the Component Diagram is divided into a series of prototypes or iterations, which are then, developed using rapid build-test cycles. The House of Quality created in the Product Design allows development to focus on quick wins, prioritizing software components that provide the highest value to the user. The process for creating the software development plan is as follows:

1. **Define the development tasks necessary for implementing the design specified in the Component diagram.** Small components should correspond to a single task, whereas large components can be broken up into several individual tasks.
   - A: Develop the Search GUI component
   - B: Develop the Content GUI component
   - C: Develop the Search Filter component
   - D: Develop the Content Filter component
E: Develop the TOPIC interface component

2. Create a dependency matrix to understand the dependencies between the tasks. Identify sets of tasks that must be implemented together. Determine the order that tasks must be implemented in.

   Table 6: Dependency Matrix of Development Tasks for the Knowledge Engineering System

<table>
<thead>
<tr>
<th>Development Task</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>B</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>C</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>X</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E</td>
</tr>
</tbody>
</table>

(E, C, A) are sequential tasks
(E, D, B) are sequential tasks

3. Separate the development plan into prototypes (short iterations), with each prototype being a complete piece of software that can be tested by the users. Use the House of Quality to map development tasks to prototypes in descending order of importance (user value).

The development plan consisted of three prototypes or iterations (Table 7). The alpha prototype focused on implementing the Search Filter component. The beta prototype added the important Content Filter tasks. The Release Candidate integrated the Search and Content Filter components into a functionally complete prototype.
### Table 7: Prototyping Plan for the Knowledge Engineering system

<table>
<thead>
<tr>
<th>Prototype</th>
<th>Tasks</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>A, C, E</td>
<td>Search Filter GUI, Search Filter, TOPIC interface</td>
</tr>
<tr>
<td>Beta</td>
<td>B, D</td>
<td>Content Filter GUI, Content Filter</td>
</tr>
<tr>
<td>Release Candidate</td>
<td>A, C, B, D, E</td>
<td>Search Filter GUI, Search Filter, Content Filter, TOPIC interface</td>
</tr>
</tbody>
</table>

4. Define a suitable software environment (programming language, framework, etc.) in which to build and test the software prototypes. For smaller systems it is generally more important to emphasize an environment that allows rapid development. For larger systems it is generally more important to emphasize performance and scalability. It is always important that the software development environment enables easy testing and deployment to users.

The prototypes were implemented using the Ruby on Rails software development environment. Ruby on Rails is a web framework built on top of the Ruby scripting language. Since the SRP was a relatively small system, we decided to use Ruby on Rails because the dynamic nature of the Ruby scripting language allowed for significantly faster development than a statically typed language such as Java or C++. In addition Ruby on Rails has built-in functionality for testing and deploying the software.

**Build and Test**

During testing the software prototype is tested by users to determine if it sufficiently satisfies their needs (the user needs). If these needs are satisfied, development proceeds to the next prototype (iteration) or deployment (if it is the last prototype). If the needs are not sufficiently satisfied then the build-test cycle is repeated.
1. Implement the individual software component(s) specified in the software plan using the detailed design specified in the Class diagrams. Integrate the software components into a working prototype.

2. Have the users actively use the prototype in the context of the work process captured in the Knowledge Engineering phase. Obtain user feedback for each user needs using a convenient scale (e.g. exceeds need, meets need, does not meet need). Record values for each technical metric in the House of Quality.

The engineers used the RC prototype while creating automation for nine different product problems ranging from relatively simple voltage alarms to complex hardware interface problems. In order to determine if the prototype sufficiently satisfied the engineer’s needs, we evaluated each user need specified in the House of Quality.

The users’ feedback after using the RC prototype was that the prototype met both of these user needs. The technical metrics “average number of pages in service request”, “average time to assess relevance”, and “average time to extract problem-solution pairs” were used to evaluate how well the SRP quantitatively satisfied these users’ needs. The results of this evaluation are described in Section 7.

5. Possible Framework Simplifications

We recommend the formal Product Design methods (Phase II) described in the framework be used whenever possible. However there are situations when it is either beneficial or necessary to simplify the application of these methods. For example, a time constraint might prohibit the use of the full framework. Similarly if the Knowledge Engineering problem is not overly complex, simplifications can be used to reduce development time and costs. The possible simplifications are listed below:

1. **House of Quality:** It is critical to identify the user needs for the product and define technical metrics for measuring these needs. In simpler or time-constrained situations it possible to omit the House of Quality and just list the user needs and technical metrics.

2. **Function Structure:** It is necessary to describe the Knowledge Engineering product in a functional form in order to ensure that the design is complete with respect to the user needs. The Function Structure can be simplified in two ways. First, adjusting the number of sub-function levels of sub-functions can control the detail of the Function Structure. Second, when the knowledge engineering problem is not complex the information flows can be omitted.
3. **Morphological Matrix**: The process of generating several possible design concepts is important in achieving a design concept that has the right balance of quality and cost. When needed, the Morphological Matrix can easily be simplified by reducing the number of solution-principles explored for each sub-function.

4. **Selection Criteria**: It is important to develop a set of Selection Criteria to assist in selecting a product concept that represents the best balance of quality and cost. However, in simpler or time-constrained situations, these Selection Criteria do not necessarily have to be formalized using the described Selection Criteria hierarchy.

5. **Utility Function**: In cases where the Selection Criteria are simple and relatively straightforward, it may be possible to select a design concept from the Morphological Matrix without formally constructing a Utility Function.

6. **Results**

In this section, we present the Service Request Portal (SRP), the Knowledge Engineering system we have developed to support the development of smart network products and services. The section is divided into three parts as follows. We begin with a description of the implementation of the SRP and how the users interact with the system. Finally, we discuss the value that the system provided from the user and organizational perspectives.

The SRP is used by engineers for assisting in identification and extraction of problem-solution pairs from customer service requests. The user interface is separated into three panes (Figure 20). The left pane controls the filtering criteria. The middle pane controls the search filter results and the right pane controls the content filter results.
Here is how the engineers use the SRP to find service requests for a specific product problem:

1. The engineer first enters a description of the product(s) of interest (e.g. “catalyst” a product line of network switch for enterprise) into the search bar located in the top pane of the user interface. This defines the high-level group of service requests that the system will return.

2. The engineer can then add optional search filters (constraints) on top of this high-level group. The results can be restricted to specific types of problems (hardware, software, etc.) using the Technology and Sub-technology fields. The results can also be restricted to specific types of problem resolutions (software configuration, replaced hardware, etc.) using the resolution code fields.

3. The SRP returns the search filter results in the middle pane of the user interface. Each search result is labeled with the service request title.

4. The engineer reads through the titles to identify possibly relevant service requests. For each possibly relevant service request, the engineer selects the result in order to obtain the service request content. The content filters in the left user interface pane define the content that the SRP will extract from the service request (summary, first and last correspondence) and display to the engineer. Once the service request is selected the content filter results (summary, first and last correspondence) appear in the right pane of the user interface.

5. The engineer reads the summary to determine if the service request is relevant. If more detail is required the user can select the “tidy” (cleaned up and no duplicates) version. If the unaltered service request is required the user can select the “original” version.
6. The engineer next reads the “tidy” version of each relevant service request to extract the problem-solution pair.

7. The engineer then codifies the problem-solution pairs into rules that can be embedded in the software of smart network products.

In order to measure the value that the SRP provided to the users, we evaluated the SRP with respect to the user needs and technical metrics captured in the House of Quality. The users were first asked to provide feedback for each user need based on their experience using the SRP. In general the users found that the SRP significantly improved their productivity when working with service requests. Users were particularly appreciative that the SRP closely integrated into their existing workflow, and therefore did not have a high learning curve. In addition they found that the content filtering functionality such as deduplication was sufficient in removing the bulk of the irrelevant content and made the service requests much easier to read. Table 8, below, shows a summary of the feedback received for each user need.
Table 8: User Needs Evaluation for the Service Request Portal

<table>
<thead>
<tr>
<th>User Need</th>
<th>Feedback</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Be able to do broad searches</td>
<td></td>
<td>Meets need</td>
</tr>
<tr>
<td>Be able to do very targeted searches</td>
<td>The tool helps to find the information easily when compared to C3 TOPIC search engine</td>
<td>Exceeds need</td>
</tr>
<tr>
<td>Be able to read service requests easier</td>
<td>It is useful to extract the information quickly</td>
<td>Exceeds need</td>
</tr>
<tr>
<td>Be able to quickly assess the relevance of a service request</td>
<td></td>
<td>Exceeds need</td>
</tr>
<tr>
<td>Be able to quickly extract problem-solution pairs</td>
<td></td>
<td>Meets need</td>
</tr>
<tr>
<td>Seamless access to service request attachments</td>
<td></td>
<td>Meets need</td>
</tr>
<tr>
<td>Easy to use</td>
<td>The portal looks just amazing. Our team will be using the portal for Syslogs IC creation</td>
<td>Exceeds need</td>
</tr>
<tr>
<td>High performance</td>
<td></td>
<td>Meets need</td>
</tr>
</tbody>
</table>

The network engineers were benchmarked, using the technical metrics from the House of Quality, during the creation of nine diagnostic rules ranging from relatively simple voltage alarms (R3) to complex hardware interface problems (R1). Two important user needs for the SRP were “be able to read service requests easier” and “be able to quickly assess the relevance of Service requests. The SRP addressed these needs by automatically generating a high-level summary of the service request. The technical metric “average number of page read to assess relevance of a service request”, was used to measure how well the SRP satisfied these. We observed that the engineers went from reading 22 pages to reading 3 pages in order to assess relevance when using the SRP (Figure 21), suggesting that the summary was sufficient for establishing the relevance of a service request in most cases.
Reducing the number of pages the engineers had to read translated to a 60% time savings when evaluating the relevance of service requests. Figure 22 shows the values recorded for the technical metric “average time spent assessing relevance of a service request”.

Once a service request was determined to be relevant, the engineers would need to perform a in-depth read to extract the problem-solution pair. Figure 23 shows the values for the technical metric “average time to extract the problem-solution pairs from the relevant service requests”. We believe that the time-savings can be attributed to the deduplication functionality that removed repeated information across
threads. Other useful features such as keyword highlighting and in allowing the users to view the attachment in browser made it easier to work the longer service requests.

Figure 123: Average Time to Extract Problem-Solution Pair from a Service Request

In order to evaluate the value that the SRP provided to the organization, we compared the increase in user productivity to the overall cost of developing the system. By using the SRP, the engineers reduced the average time spent assessing the relevance of each service request by 60% (15 minutes). The time to extract the problem-solution pair was also reduced by 30% (30 minutes) per a service request. Each engineer read on average approximately ten service requests daily and extracted problem-solution pairs from two of these service requests. We estimate the impact of the SRP for the ten engineers to be on the order of 35 hours timesavings every day.

We next consider the overall cost of the system. The user-centric approach allowed us to focus on the functionality most important to the users, allowing the prototypes to be developed rapidly and at relatively low cost. The prototypes were developed by two developers working over a six-month period (approximately 1500 man hours). Open source software was used, so the only significant development cost that needs to be considered is the developer time. Assuming that the developer time and engineer’s
time are equally valuable, the break-even point for this system is around one and a half months (43 days). We therefore conclude that the system was cost effective and provided value to the organization.

7. Conclusion and Future Work
In this paper we have presented an integrated framework that combines methods and techniques from the domains of Knowledge Engineering, Product Design, and Software Engineering for developing high-value Knowledge Engineering systems. Our novel contribution is the use of methods and techniques from Product Design in order to ensure that the developed system sufficiently addresses the users’ needs and represents the best value (quality and cost). We have demonstrated the framework to have good results on a simple but non-trivial Knowledge Engineering problem within the domain of computer networks.

The proposed framework should prove to be valuable to high-technology organizations interested in developing Knowledge Engineering systems. The framework provides a structured set of tools for resolving typical trade-off conflicts that arise in the design, development, and deployment of a Knowledge Engineering system. Based on our experiences, we have found that the framework can simplify the work in Knowledge and Software Engineering while substantially increasing the quality and cost effectiveness of the system.

Framework refinement to be addressed in future work include: expanding the Knowledge Engineering phase to include more sophisticated Data Mining and Information Retrieval techniques and including other useful UML models such as the Sequence diagram in the Software Engineering. The framework also needs to be further tested on wider range of Knowledge Engineering problems.

8. Acknowledgements
The authors would like to acknowledge the help of Daniel Core in earlier versions of this paper. In particular Daniel made useful contributions to the user needs, function structures, and also implemented
the GUI module of the KE System from the case study. We would also like to acknowledge the support of the Cisco Network Management Operations Lab and in particular the support of Sri Ramachandran of Cisco, and Professors Pat Mantey and Brad Smith of UCSC.

References


