**A Computational Framework for Authoring and Searching Product Design Specifications**

Alexander Weissman, Martin Petrov, and Satyandra K. Gupta
University of Maryland, College Park

Xenia Fiorentini, Rachuri Sudarsan, and Ram Sriram
NIST

# Abstract

The development of Product Design Specifications (PDS) is an important part of the product development process. Incompleteness, ambiguity, or inconsistency in the PDS can lead to problems during the design process and may require unnecessary design iterations. This generally results in increased design time and cost. Currently, in many organizations, PDS are written using word processors. Since documents written by different authors can be inconsistent in style and word choice, it is difficult to automatically search for specific requirements. Moreover, this approach does not allow the possibility of automated design verification and validation against the design requirements and specifications.

In this paper, we present a computational framework and a software tool based on this framework for writing, annotating, and searching computer-interpretable PDS. Our approach allows authors to write requirement statements in natural language to be consistent with the existing authoring practice. However, using mathematical expressions, keywords from predefined taxonomies, and other metadata the author of PDS can then annotate different parts of the requirement statements. This approach provides unambiguous meaning to the information contained in PDS, and helps to eliminate mistakes later in the process when designers must interpret requirements. Our approach also enables users to construct a new PDS document from the results of the search for requirements of similar devices and in similar contexts. This capability speeds up the process of creating PDS and helps authors write more detailed documents by utilizing previous, well written PDS documents. Our approach also enables checking for internal inconsistencies in the requirement statements.

# 1 Introduction

The process of developing a new electro-mechanical device generally begins with generating a product design specification (PDS) document[1]. The PDS document describes the intended function of the device being designed, and the environment in which it will be used. This information is usually expressed as a set of requirement statements, which outline the design goals, constraints, and the intended device behavior. A requirement statement is an English sentence that expresses a rule with which the final design solution must comply [1]. For example, "The product must be able to function underwater" is a requirement statement.

In addition to intended functionality, a PDS may also include statements which describe information on how the device will be marketed, produced, and distributed, how the device should be maintained and disposed of, and the standards and regulations that apply to each of these areas. Several definitions of PDS exist. We have included two representative definitions of

[1] In this paper we use the terms PDS document and requirements document interchangeably.

PDS. According to Magrab [2], "the PDS contains all the facts relating to a product's outcome. It is a statement of what the product has to do and is the fundamental control mechanism and basic reference source for the entire product development activity." Dieter [3] states that "The product design specification (PDS) is the basic control and reference document for the design and manufacture of the product. The PDS is a document which contains all of the facts related to the outcome of the product development." We can infer from either of these definitions that all decisions made during the design process must be carefully made with respect to the PDS.

A well-written PDS document is design solution neutral and does not specify design details; i.e., it describes what the product should do and not how it does it. This is crucial to ensure that the creative control of the designers is not stifled, and that changes to the design details will not necessarily require a change to the PDS. Furthermore, with regard to communication within large design teams, the PDS serves to ensure that every member of the team is working towards the same overall goals [4].

Once the PDS has been completed, the engineering design team can begin work on a solution, which meets all of the requirements and specifications described in the PDS. To create a detailed model of the product, concepts are generated, refined, and elaborated until the final design is determined. This detailed model includes information about the components and assembly and how they function together to realize the stipulations provided by the PDS.

An abridged example of a traditional PDS is given in Figure 1 [2].

<table>
<tr>
<td>
Performance<br>
• Tape inside corners.<br>
• Tape joints in any orientation.<br>
• Tape joints without leaking joint compound.<br>
• Taped joints will require no additional smoothing.<br>
• Tape will not break prematurely.<br>
Shipping<br>
• Will be shock, vibration, and weather resistant for shipping by any means: vibration environment from 4-33Hz at 0.06 inch (1.5 mm) amplitude; shock environment simulated with an 8 foot (2.4 m) drop test. Insensitive to temperature and humidity.<br>
• Will be packaged in a rectangular cardboard box that can be stacked up to 8 ft (2.5 m.)<br>
Aesthetics<br>
• Product will have a durable finish.<br>
• Colors will be green and black.<br>
• Product will convey ruggedness.
</td>
<td>
Maintenance<br>
• All fasteners will be standard.<br>
• All components subject to wear to be inexpensively and easily replaceable with standard tools and no special skills.<br>
• Design will be modular for easy repair and cleaning.<br>
• Joint compound removed after each day's work.<br>
• No or very few lubrication points.<br>
Disposal and Recycling<br>
• Product will not contain any environmentally hazardous materials.<br>
• Easily disassembled for component recycling and reuse.<br>
Product Environment<br>
• Product must operate in the following environment:<br>
– Temperatures between 40 and 120°F (4 and 50°C)<br>
– Atmospheric pressure from sea level to 7,000 ft (2.1 km)<br>
– Relative humidity to 100%<br>
– Concentration of solid or liquid particulate smaller than 500μm.
</td>
</tr>
</table>

**Figure 1: Sample requirements statements from PDS for Drywall Taping System [2]**

Unfortunately, there are a number of problems with the current paradigm for PDS authoring. The first problem is that although requirements must convey information about the design in a precise, unambiguous manner, they are written in natural language. Statements written in natural language can be ambiguous unless additional information is provided on how to interpret the document. A simple example is the phrase "clamp mechanism." In this phrase, it

is unclear whether the word "clamp" is a modifier to describe the type of mechanism, or a verb to indicate performing the action of clamping on the mechanism. In requirement statements, ambiguity in the meaning or context of a word can lead to costly design errors.

The second problem is that authoring a PDS document for a new product requires significant time and expertise. When designing a new product, the author of the PDS has two choices; writing the PDS from scratch, which requires a detailed level of knowledge about how the product should behave within a wide range of technical, economic, and social domains, or adapting requirement statements from prior PDS documents. Reusing statements in a previously written similar PDS is certainly a possible approach to ensuring completeness and reducing the time needed to write PDS. This strategy is particularly common when it comes to very generic requirements, which become part of PDS authors' repertoire after much experience [5]. This approach, however, is error-prone and can become time consuming if PDS documents are stored as text documents. The author may not find all relevant requirements, fail to properly adapt an imported requirement for the new product, or improperly include a requirement that does not apply to the new product. Reusing previous PDS is also difficult since the author needs to understand the different design context in which the requirements were developed. Incompleteness, ambiguity, or inconsistency in the PDS can lead to problems during the design process and may require unnecessary design iterations and increase the design time and cost. Hence, an improved approach is needed to write explicit, unambiguous PDS quickly. This approach should enable fast searching of previously written PDS documents for relevant, reusable requirements.

Finally, PDS often suffer from consistency and redundancy issues, which can confuse or mislead designers, wasting time or leading to mistakes. For example, a PDS might contain two requirements, one which states that the "mass of the product should be less than 50 kg," and another which states that "the mass of the product should be between 40 and 60 kg." It is not clear whether the designer should consider the intersection of these two constraints (40-50 kg), or whether one of these statements is in error and was perhaps left in from a previous version of the PDS.

To address these problems, we present a computational framework and a software tool for writing and searching computer-interpretable PDS. To improve searchability and reduce ambiguity of requirements, our framework introduces a means for assigning specific meaning to different portions of the requirement statements. Requirement statements are written in PDS authors' own words, but important words and phrases are defined in terms of quantitative values, keywords from predefined taxonomies of unique, strictly defined terms, and other metadata. This approach is important for both computer-based search tools and human authors, because it represents quantitative values in a consistent manner, and ensures that the semantic usage of a given word is consistent from document to document.

To address problems associated with PDS authoring and requirement reuse, our framework organizes the PDS based on the stages of the product's life. Every requirement statement has bearing on some stage of the product's life, whether it is marketing, manufacturing, usage, or disposal. Thus, these stages provide a good basis for requirements authoring, which can be broken down to a sufficiently fine level of granularity to be useful to the author. This breakdown forms a set of categories in which the requirement statements reside. Requirements statements are written by thinking through the various life stages of the product and determining the appropriate requirements at each stage. Each category also contains a

number of objects, which are physical or abstract entities that interact with the device being designed. Using our framework, PDS authors can associate categorized requirement statements with interaction objects within those categories.

PDS documents written in this manner can then be searched automatically based on the relevant interaction objects in the new PDS. For example, if the author knows that the new product will be shipped by truck, a search could be performed on all requirements in the "Shipping" category that are associated with the object "truck." Furthermore, our approach supports searching on the relationships within statements between the device and interaction objects by modeling verbs and grammatical subject/object relationships.

Finally, our framework provides for consistency checking of mathematical expressions by explicitly representing them in terms of attributes of the devices or interaction objects. Software which checks these relationships to verify consistency can then be developed.

The details of the framework are explained in Section 3. To support authoring and searching of PDS documents, our approach is designed to be both human-usable and computer-interpretable. To accommodate the need of continuously refining and improving the keyword taxonomies, the framework has also been constructed to be customizable and extensible.

Section 4 describes the results of case studies involving five PDS documents written by students in a senior design class at the University of Maryland. Analysis is performed with regard to the framework's ability to capture all essential information from PDS and consistently map terms in a typical PDS, and the subsequent need to rewrite requirement statements to fit the framework. This analysis is crucial for establishing the usability of our framework for authoring PDS in general.

Section 5 describes a software tool based on the ideas described in previous sections to aid PDS authors in categorizing and labeling their requirement statements. The entire PDS, including categories, statements, and mappings of requirement statements, is stored in the XML data format. The tool includes features to assist authors in finding appropriate taxonomy words for their requirement statements, and to generate PDS in HTML format, suitable for inclusion in professional reports.

PDS documents are written in very different styles by different people. Section 6 provides an example of how PDS documents represented using our framework can be quickly searched using advanced queries that go beyond a simple keyword search. Being able to perform these queries automatically on large databases of existing PDS documents will enable authors to easily adapt and reuse a significant number of requirement statements for new projects.

## 2 Literature review

The AeroSpace and Defence Industries Association of Europe proposes - in the ASD-STE100 [6] standard - to adopt a controlled form of English called Simplified Technical English to write technical documentation. ASD-STE100 was originally developed to avoid lexical ambiguity and sentence complexity in aircraft maintenance documents. Today, it is used in many other industries and for all kinds of technical documentation including PDS. The Simplified Technical English is composed of a dictionary of controlled vocabulary and a set of writing rules.

In the STE dictionary, all synonyms words are grouped together and only one word can be used to express them. Moreover, for each word a unique and precise definition is given, together with the part of the speech it can represent. As an example, according to the STE specification: 1) the words "dull" and "faint" should be avoided and replaced with the word "dim;" 2) the word "dim" can be used only as an adjective and its meaning is "not bright," rather than "hopeless;" 3) using the word "dim" as a verb is prohibited: the verb "decrease" should be used instead. The usage of this controlled vocabulary is regulated by "writing rules." The goal of these rules is to help readers to understand a technical document. The writing rules impose, for example, usage of the active form of verbs when possible, prohibition of noun clusters of more than three nouns, a limitation of at most 25 words in each sentence, and expression of only one topic in each sentence. The writing rules also provide some indications on how to integrate proprietary taxonomies (called Technical Names and Technical Verbs) into the controlled vocabulary. For this purpose, they list all the technical categories to which the proprietary words should belong.

Several software tools have been developed to assist authors to write technical documents in simplified English conforming to ASD-STE100. Unfortunately, these tools and software can only check simple rules, such as sentence lengths and noun clusters, but they cannot check other rules, such as number of topics in each sentence and content meaning. As declared by the ASD association, even with the help of these tools, "training is the first essential step for a technical author to be able to apply ASD-STE100 correctly" [7]. To adopt the ASD-STE100 standard, companies are thus required to devote a considerable amount of time and resources to training.

The ASD-STE100 standard imposes a standard list of words with predefined meanings for use in requirement statements while leaving the user free to choose the statement structure. Our approach goes beyond that by giving the user flexibility in choosing the word used to write the statement, but then mapping those words to keywords in a predefined taxonomy. Our framework also supports explicit representation of mathematical relationships.

One possibility would be to simply represent PDS as they are currently represented in plain text, and then implement semantic search capabilities through a natural language processor (NLP). However, this approach presents many challenges because natural language processing is not perfect, and significant time would be consumed by double-checking the results from the NLP. Natural language processing presents computational challenges in dealing with phrases and sentence fragments. Unfortunately, many requirements are currently written in this style. Finally, using NLP in search would not help to elicit additional information during the authoring process. Our framework helps to encourage authors to more precisely define inherently imprecise terms such as "hot," or "fast" in terms of explicit quantities (temperature, speed).

Zeng and Chen [8-12] propose some concepts, which we have adapted for our model, such as defining a product in terms of the elements of its environment, the use of requirement categories based on the product's life stages, and mapping natural language to a standardized representation. However, they do not provide an implementation strategy for actually authoring a PDS. Other authors have discussed ideas such as requirement reuse [13], taxonomic classification of requirements [14, 15], and the duality between "real-life" requirements and formal requirements [16]. Graphical representations of requirements have also been proposed [17].

In comparison with software engineering, there have been limited efforts towards any standardized representation of requirements in the field of electro-mechanical products. Models for software requirements are well-developed [18-24]. However, these models are specialized and cannot be applied to the electro-mechanical domain without significant extensions.

Currently, most work on requirements representation for electro-mechanical products focuses on requirements management [25-27]. Requirements management techniques seek to first analyze the processes of requirements generation and propagation and then provide constraints and rules for these processes with the goal of guaranteeing ownership, prioritization, agreement, and communication of the requirements. However, they do not attempt to provide constraints and rules for the actual PDS document. To date, most efforts have largely been carried out within the field of system engineering: the requirements are generated at the level of product system and subsequently refined for product design. In this section, we provide a brief overview of these efforts.

The EIA-632 [28] standard describes two fundamental processes for system engineering: requirements definition and solution definition. In the requirements definition process, system technical requirements are derived from stakeholders' requirements. In the solution definition process, a logical and physical representation of the system solution is presented and system technical requirements are refined at the subsystems level. EIA-632 defines requirements as "Something that governs what, how well, and under what conditions a product will achieve a given purpose" and classifies them into operational, performance and enabling requirements. Operational requirements focus on the goals, objectives, and general desired capabilities of the system without indicating how the system can be implemented. Performance requirements focus on how well the system is suited to perform a function, along with the conditions under which the function is performed. Enabling requirements include technology constraints, product design constraints, and requirements associated with the processes of the product lifecycle. The EIA-632 also lists the desirable characteristics of each requirement statement: clarity, correctness, feasibility, focus, implementability, modifiability, certainty, singularity, testability, and verifiability. This standard provides a definition for each of these characteristics but it was outside the scope of this standard to suggest any means to achieve them.

Similarly, the IEEE 15288 [29] standard presents a framework for describing the system lifecycle processes. This standard recommends a process for requirements derivation that is similar to EIA-632. First, stakeholders' requirements are translated into system requirements. Second, the system is characterized with its functions, the performance of its functions and the conditions under which the functions will be performed. Third, the system architecture is developed and the subsystems requirements are specified. Once again, the IEEE 15288 lists the characteristics required for each requirement statement but it does not describe how to achieve them.

The landscape of the standards within the field of systems engineering includes not only process standards (such as IEA-632 and IEEE 15288) but also modeling standards [30]. The System Modeling Language (SysML) [31] is a language that provides a means to capture the system modeling information, which includes system requirements. The requirements categories proposed in SysML closely follow the other standards: requirements are divided into functional, interface, performance, physical and design constraint categories. In SysML, the requirements are expressed in plain text and simply connected to the system model in a requirements diagram. In SysML, requirements diagrams, requirements, design elements, and test cases are connected

to each other through relational stereotypes (derive, satisfy, verify, refine, trace, copy, and contain). SysML provides a text-based definition for each of these stereotypes. Although SysML allows tracing of relationships between requirements and design elements, the usage of plain text to express requirements remains one of the major drawbacks of SysML. The framework we propose in this paper could extend SysML by replacing the plain-text representation of requirement statements with statements categorized and defined using our keyword taxonomies, mathematical expressions, and other metadata.

The Core Product Model (CPM) and Open Assembly Model (OAM) were created at the National Institute of Standards and Technology (NIST) to capture the information related to the full spectrum of product development activities. CPM focuses on an artifact representation that encompasses a range of engineering design concepts beyond the artifact's geometry, including function, form, material and requirement; as well as physical and functional decompositions, mappings between function and requirement, and various kinds of relationships among these concepts [32]. OAM extends CPM to include the representation of assembly products and the relationships between products components [33]. Similar to SysML, CPM allows tracing of relationship between requirements and functions, form, material, and geometry, and can be extended to include the requirements statements themselves.

Existing software tools for requirements engineering include commercially available software such as DOORS [26], CORE [34], and Teamcenter Requirements (SLATE) [35], and tools based on the Resource Description Framework (RDF) such as the XML/RDF Traceability Viewer [36].

DOORS is a collaborative requirements management tool which defines relationships among and between product requirements and design solution elements. This establishes a traceable network which can be used for design validation and verification by applying appropriate reasoning engines. Design bottlenecks and critical components can thus be identified.

The CORE tool set is sold by Vitech and forms part of a model based system engineering methodology (MBSE). MBSE methodologies are defined as a collection of processes, methods, and tools that can be applied to particular classes of systems engineering problems [37]. This methodology defines four design domains: requirements, function/behavior, architecture, and validation/verification. These domains are linked through a common System Design Repository, which provides a common, structured representation which can define attributes for and relationships among requirements, functions, components.

Teamcenter Requirements by Siemens, formerly known as SLATE, integrates system engineering and requirements management to provide enterprise level collaboration capabilities. Requirements can be imported from Word documents, which Teamcenter searches for keywords from a standard database. Based on these keywords, requirements are then placed into an appropriate category. As design representations can also be represented in Teamcenter, traceability is also established through keyword matching.

The Resource Description Framework (RDF) [38] is a widely used framework which builds upon the Extensible Markup Language (XML) by establishing subject-predicate-object relationships between resources. In the scope of requirements engineering, resources may include requirements, design concepts and elements, stakeholders, attributes, and elements of a product's environment. The XML/RDF Traceability Viewer developed by Scott Selberg uses

RDF to create hierarchical relationships among requirements as a "refines/is refined by" relationship, and between requirements and design elements as "satisfies/is satisfied by" relationships. These relationships can then be visualized as trees or block diagrams.Together with reasoning engines and suitable ontologies represented in a standard such as the Web Ontology Language (OWL) [39], designs can be verified against their requirements. It is hoped that the framework presented in this paper can be used to further enrich the RDF representations of the requirements, and the resulting traceability support. Unlike DOORS, CORE, Teamcenter, and the Traceability Viewer, our framework establishes relationships not just among requirements and design elements, but within requirements, among the constituent phrases. Furthermore, the taxonomies of terms with established meanings and relationships constitutes an ontology, which could be represented in OWL. For simplicity, however, the sample taxonomies and PDS documents presented in this paper are represented in plain XML.

# 3 A Framework for Representing the Product Design Specification

## 3.1 Overview

The design of our framework brings together concepts from the different parts the design community. Several different categorization schemes for requirement statements have been published in design engineering textbooks [1, 2, 40]. Our framework merges these schemes into a detailed set of categories and subcategories based on the stages of the product's life, from initial design through manufacturing, usage, and disposal. Standard taxonomies of unambiguously defined keywords have been proposed for SysML [31], as well as for Functional Bases and Functional Definition Templates [41]. We have adapted these taxonomies to fit the specific needs of requirements representation. The syntax for representing mathematical constructs is drawn from Matlab®, a well known and widely used numerical computing environment. These concepts have been combined and modified to produce a robust framework for representing PDS, which makes it easier to search for relevant requirement statements and apply them in new documents.

With the above in mind, we have identified four main goals for the PDS framework, which together aim to make it easier to author a PDS, and to improve the results of automated search patterns.

- ***Usability***: The framework should not impose constraints on how the requirement statement is initially written. Imposing constraints would require the author to conform to a rigid and unnatural way of writing, slowing down the authoring process. Instead, it should simply propose a toolbox for categorizing and defining traditionally written requirement statements.
- ***Customizability***: The framework should be modular to allow users to discard some portions of the framework and replace them with their own.
- ***Extendibility***: Similarly, users should be able to extend or refine parts of the framework in a controlled fashion.
- ***Computer-Interpretability***: Statements represented using the framework should be able to be consistently processed in a reliable and automated way. This is crucial for developing computerized tools such as search engines and design validation software.

In Section 4, we demonstrate that our framework meets these four goals outlined above.

Our framework for PDS authoring consists of five steps:

1. the device being designed is identified and defined
2. relevant instantiations of environment objects are declared
3. requirement statements are written by stepping through the requirement categories and composing or importing appropriate requirement statements for each stage of the product's life
4. statements are broken into phrases by the author or an automated natural language processing tool
5. phrases are defined in terms of keywords from attribute and verb taxonomies, instantiations of the interaction object taxonomy, mathematical constructs, and other metadata.

The framework and the relevant components used in each step are shown in Figure 2. We now describe each step of the framework and define relevant components for the step.

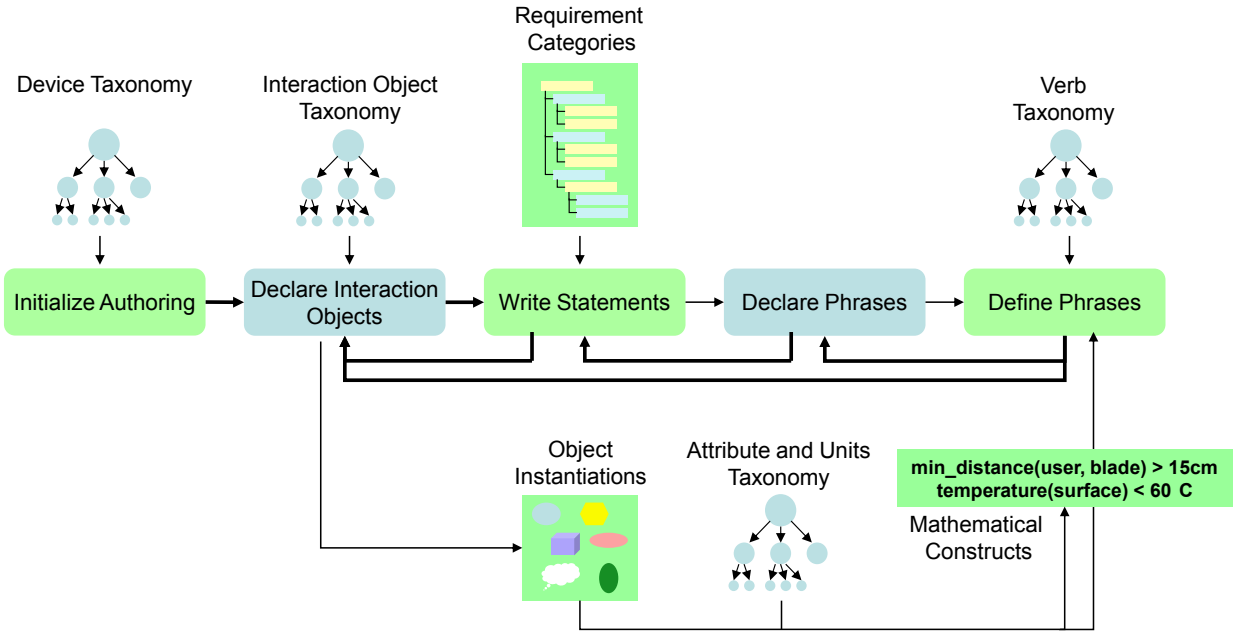


**Figure 2: Process and relevant components for authoring a PDS document.**

## 3.2 Initialization of the Framework

Authoring of a product design specification begins with identification of the device being designed. A unique label for the device is chosen, and mapped to a keyword in the device taxonomy. Additional data, such as author names and version numbers, can be part of this definition. The device taxonomy is a set of predefined, hierarchically associated devices. Representative keywords, which name the devices, are classified according to certain criteria, such as their primary working principle. The taxonomy is represented as a tree, and devices with similar working principles are grouped as siblings on the tree.. Since device taxonomies are

specific to a very small subset of engineering design, each company or organization may ultimately develop its own, customized device taxonomy. However, different device taxonomies from different sources may be aggregated or harmonized as the model becomes more widely adopted, for example in the form of a public Internet database or reference data libraries as adopted in STEP AP239 [42]. Parts libraries such as PLIB [43] may also be a source of device taxonomies. Our framework can easily support importing of PLIB into our internal taxonomy structure. Once a taxonomy has been built for a class of devices, the elements of the taxonomy can be used to map references to the product in every requirement statement [44].

A sample device taxonomy for kitchen appliances, based on their primary working principles (the means by which they accomplish their function), is shown in Figure 3. The working principle criterion was chosen because most appliances have only one or two primary working principles (e.g. cooking and mixing), which thus minimizes the number of mappings necessary to categorize a given device name. As an example of how this taxonomy can be useful, consider the device "Automatic Pot Stirrer," an appliance that is meant to simultaneously stir and cook food. Virtually every requirement statement for the "Automatic Pot Stirrer" will contain a reference to this device, so it would be useful to associate some standard terms with the device in each statement. In this case, appropriate terms would be "Device/Mixer" and "Device/Cooker/Water Cooker" or "Device/Cooker/Conduction Cooker." This connection is extremely useful for comparing the device with other products that might have a similar purpose but a different name. No matter how the specific instance of the product is labeled (e.g. "Automatic Pot Stirrer" vs. "Mix-n-Cook"); it will always be linked to one or more of these labels in the device taxonomy.

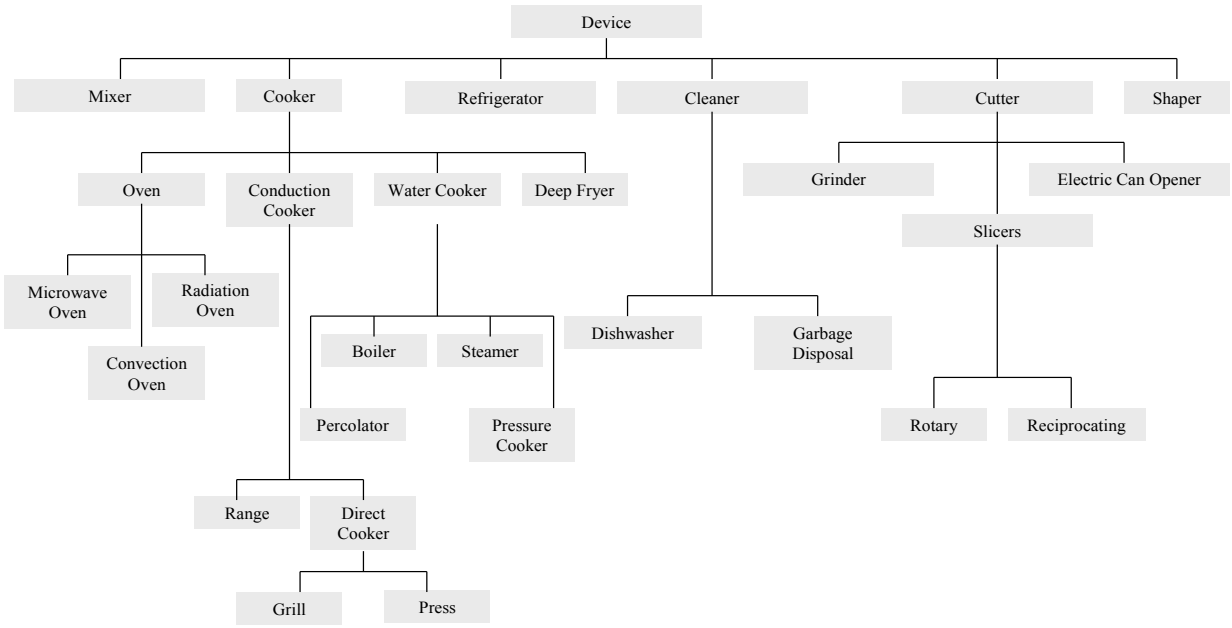


**Figure 3: A sample taxonomy of kitchen appliances based on primary working principle.**

## 3.3 Instantiation of Objects that Interact with the Device Being Designed

For each type of device, such as kitchen appliances, there is an associated environment. Kitchen appliances are used primarily in the kitchen, automobiles are used on roadways, and

battle tanks are used in combat zones. It therefore makes sense to form a standard taxonomy of objects for the environment in which the product exists [8]. These objects can then be instantiated and used to define portions of requirement statements. A basic interaction object taxonomy has been developed and is included in the PDS Author software tool. PDS authors can add additional, device-specific objects to this taxonomy as necessary. This taxonomy follows the same structure as the statement categories presented in Section 3.4; under each requirement category, appropriate environment objects are placed. By associating objects with categories, search tools can be used to determine which requirements in a given category are applicable to a new design, based on which objects appear in the new product's environment.

In the same way as the device is defined, interaction objects are instantiated by declaring a unique name for the object, and then mapping them to words in the interaction object taxonomy. Just as with device taxonomies, object taxonomies can be aggregated into larger hierarchies. Sample object taxonomy for a few representative stages of the environment for a kitchen device is given in Figure 4.

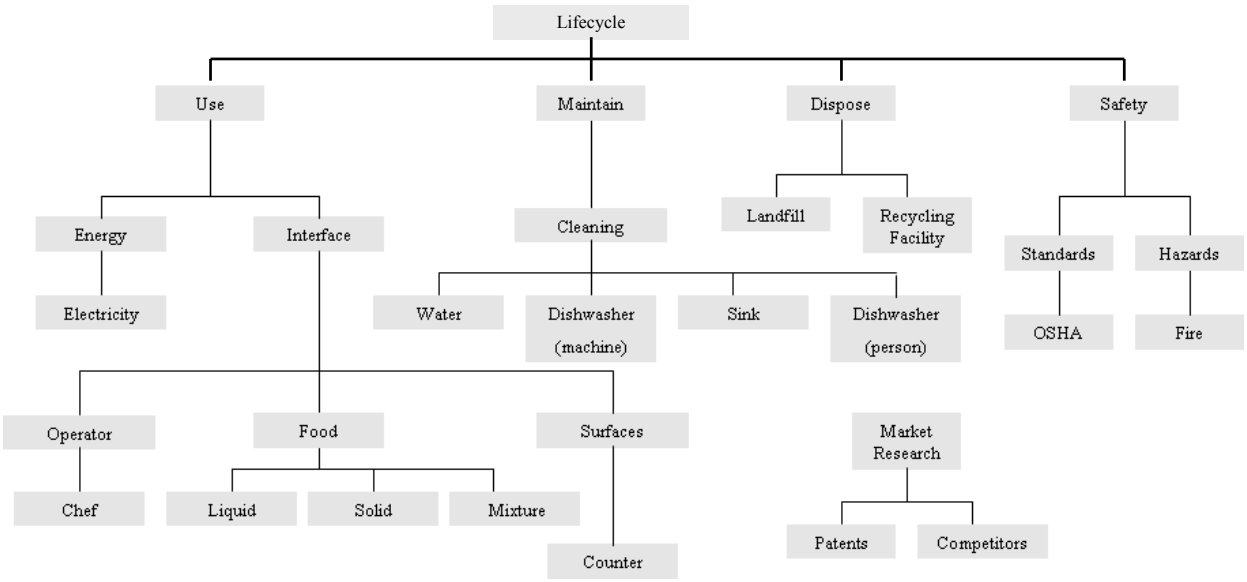


**Figure 4: A sample taxonomy of interaction objects typically encountered by kitchen appliances.**

## 3.4 Writing Requirement Statements

Once interaction objects have been declared, the author begins writing the requirement statements. The initial representation structure consists of empty categories, which have been chosen to reflect the product life stages. The author fills the relevant categories with appropriate requirement statements, importing and modifying requirements from previous PDS documents when applicable, and composing new statements when necessary.

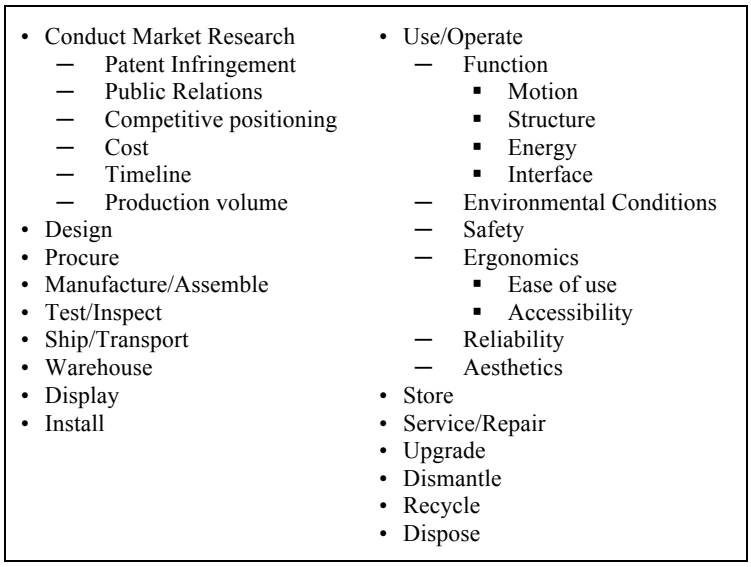


**Figure 5: Requirement categories for product life stages.**

Our approach allows the author to assign each statement to one or more categories based on its content. A hierarchy of categories has been constructed based on the product life stages, which is shown in Figure 5. Many of the subcategory names, such as "Cost" and "Ergonomics" have been taken from the categories given in Pahl and Beitz [40]. Other categorization schemes [1, 2] have been suggested, but none of these cover the environment of the product for its entire life.

This organization is useful, for example, for a development team member who is only concerned with a specific aspect of the product's life. For example, a marketing manager may want to reference the aspects of the PDS that are relevant to marketing the product, without seeing the details on the manufacturing constraints. Furthermore, it helps to ensure completeness by forcing the author to consider product requirements at all stages, rather than just cost, manufacturing, and usage.

Notice that requirement categories are represented as a hierarchical tree structure. This is useful for organizing requirements based on how closely they are related to one another [45]. For example, according to the hierarchy given in Figure 5, a statement mapped to the category "Structure" is more closely related to a statement mapped to "Interface" than one mapped to "Cost." Authors may also extend the requirement categories in Figure 5 with their own sub-categories when a finer distinction is required.

## 3.5 Declaration of Phrases in Requirement Statements

The written representation of a piece of information in natural English does not always fully capture the meaning. For example, a word's meaning may not be completely clear without considering the context of the phrase, sentence, paragraph, or document in which it resides. As discussed in Section 2, various controlled languages have been developed which seek to establish a stricter correspondence between the syntactic form of a statement and its meaning. Unfortunately, none of these gives a level of expressiveness suitable for daily use in the broader field of mechanical design. Therefore, our goal for this part of the framework is to reinforce the

connection between the syntactic form of a statement and its meaning, while preserving expressiveness.

To achieve this goal, we allow the author to parse requirement statements into phrases, and then define them. These defined phrases are based on case grammar theory [46], which models a sentence in terms of its core verbs and the phrases related by the verb, called deep cases. In case grammar theory, verbs are analogous to functions in computer programming. Like a function, a verb can have zero or more arguments, each of which has a specific type. These arguments, instead of data structures, are grammatical phrases, which are typed by their semantic role in a sentence. For example, the verb "rotate" has two arguments. The first argument has the type "Agent," which is the subject of the verb. The second argument has the type "Object," which is the object of the verb. A valid phrase for the "Agent" argument might be "the user," and a valid phrase for the "Object" argument might be "the doorknob." Verbs can also have optional arguments and, like programmatic functions, can be overloaded to support usage of the verb in different senses. For example, the same verb "rotate" also has an intransitive case. In this case, it has only an "Object." A valid phrase for the "Object" argument in this case might be "The Earth."

In our framework, we greatly simplify the semantic distinctions of the deep cases used in linguistic circles. Once again, this is done for the convenience of the PDS author, and to decrease the learning curve necessary to use our framework. PDS authors, after writing a requirement statement, may then identify the verb and its deep case arguments by labeling them using the phrase types we present below. Natural language processing tools may also be useful in assisting the author by automatically parsing statements, breaking them down into phrases, and identifying the phrase types [47].

The **subject-phrase** is used to identify the subject of a statement, which is usually the first phrase of the sentence. This is somewhat analogous to the "Agent" deep case in case grammar theory. "The user," "The gear mechanism," "The color of the surface," and "Use of a DC power source" are all valid subject phrases.

A **verb-phrase** consists of a core verb, plus any auxiliary verbs, negations, and infinitive or participle words. For example, "operate," "not be operated," "be able to operate," "be able to be operated," and "be able to continue operating" are all valid verb-phrases. The definition for a verb-phrase includes the core verb, the inflection of the verb, such as transitivity, potentiality, and negation, and references to the phrases, which constitute its arguments. The arguments of a verb must be consistent in valence (number of arguments) and argument type with the valence and types specified in the verb taxonomy.

Our proposed core verb taxonomy is adapted from Ira Golden's thesis [41], which contains 30 generalized hypernyms and 827 hyponyms, or more specific synonyms, below each hypernym. Our taxonomy has been constructed from Golden's taxonomy by aggregating those verbs in the taxonomy, which are defined as semantically equivalent according to synonym sets in Wordnet [48]. Wordnet is a lexical database which groups nouns, verbs, adverbs, and adjectives into synonym sets ("synsets"), and then links these synsets to one another according to their semantic and lexical relationships. The preferred word from each synset is placed in our taxonomy, and all other synonyms are discarded. The reasons for this strategy are clarified in Section 5. The words in our taxonomy are then arranged hierarchically based on similarity. For each verb, we also keep track of the argument valence and type. Some verbs are represented

multiple times, in different senses with different argument valences and types. The full taxonomy can be seen in the NISTIR report [49].

**Object-phrases** include **what-phrases** and **who-phrases**. These phrases are used to define a thing or person acted upon or characterized by a verb. The distinction between what-phrases and who-phrases is determined by whether the object is inanimate or animate. An example what-phrase might be "between 10 and 20 kilojoules" or "cloudy days." An example who-phrase might be "the operator."

Modifying phrases are used to augment or specify more details for other phrases in a statement. A modifying phrase can be a **when-phrase**, **where-phrase**, **what-state-phrase, how-phrase**, or **why-phrase**. **When-phrases** can be used to specify a definite or indefinite period of time, or absolute date such as "2-3 years," "after January 14th," or "in approximately 18 seconds." **Where-phrases** specify a definite or indefinite location such as "next to the electrical outlet" or "10 cm from the inlet valve." **What-state-phrases** indicate some other condition that cannot be represented as a time, as with a when-phrase, or a physical place, as with a where-phrase. "Under normal loads" and "without interference" are examples of what-state-phrases. When-phrases, where-phrases, and what-state-phrases are analogous to the deep case "Location."

**Why-phrases** are used to characterize purpose within a statement. For example, "in order to provide power" and "to simulate five years service" are valid why-phrases. Why-phrases are analogous to the "Benefactor" deep case.

**How-phrases**, such as "by increasing the hydraulic pressure," indicate the manner in which an action is to be performed, or a means by which the action is to be performed. They are similar to why-phrases but emphasize method rather than purpose. As mentioned earlier, the PDS should not propose specific design elements, which would satisfy the requirement. Thus, the how-phrase must not be used in this manner. Rather, the how-phrase may be used to specify some general constraint on the action being performed.

Phrases can also be broken down hierarchically; a phrase can be defined as a certain type and then broken down into multiple sub-phrases, which each have their own definitions. For example, the statement "The device shall allow the user to adjust the volume" can be broken down into the subject-phrase "the device," the verb-phrase "allow," and the what-phrase "the user to adjust the volume." This what-phrase, in turn, can be broken down into the following subphrases: (1) the subject-phrase "the user," (2) the verb phrase "to adjust," and (3) the what-phrase "the volume."

Defining phrases using this case frame approach is extremely useful for searching for words, which are related to one another as typed arguments of a verb. In addition to searching for keywords in a requirement statement, the user can also specify which words should appear in the verb of the statement, and which words should be contained in the verb's specific arguments. For example, one could search for requirements in which a piston is supposed to drive a crankshaft. By specifying "drive" in its transitive sense – Drive (subject-phrase, what-phrase) – and by specifying that the word "piston" should appear in the subject-phrase and the word "crankshaft" should appear in the what-phrase, the search rules out statements in which, for example, the crankshaft drives the piston. It also rules out statements in which the verb "drive" is used in a different sense, such as in "a piston and crankshaft allow the car to drive well." Additional examples of how these types of searches might be used are presented in Section 6.

Once the phrases and their organization within a statement have been identified, software tools can be developed which match statements based on structural patterns rather than on specific words. Not all parts of a requirement statement need be defined; in fact, the author could choose to leave one or more entire statements undefined. However, the computer-interpretability of the statement, and therefore the advantages provided by our framework, improve as more phrases are defined.

## 3.6 Phrase Definitions

After parsing their statements into typed phrases by identifying core verbs and their arguments, the author defines each phrase. The definition may include identifying key verbs, nouns, adjectives, prepositions, and adverbs, conjunctions, interjections, and articles using keyword taxonomies, mathematical constructs, and other metadata. In this manner, phrases can be semantically defined by the user in precise, unambiguous terms, and reused throughout the document.

To define certain keywords, authors can map their own terminology to one or more words in a standard taxonomy in which each word has a precise, predetermined, and unique definition. Therefore, the meaning of each statement can be conveyed using our representation without ambiguity in the form of synonyms and homographs. This taxonomy constitutes a lexicon, which is common to all requirements documents within a particular domain. Words, which have little value in keyword searches, such as prepositions and adverbs, do not have associated taxonomies.

We have already discussed device, verb, and interaction object taxonomies and their advantages; they allow non-standard vocabulary to be used by the PDS author, and yet remain linked with a standard lexicon. Our framework also includes taxonomy of the attributes of devices and objects, and their associated units. Attributes are keywords, which represent properties of devices and objects. Although many attributes (e.g. Reynold's number) are relevant to only a handful of devices, other attributes such as mass, length, and cost are applicable to virtually any device. Therefore, for the purpose of providing a simple, consistent lexicon of attributes, we declare every attribute global as applied to all devices and objects. Associating attributes with requirements has also been used as a way to perform evaluation tasks [50]. Based on the 'property' taxonomy in Ira Golden's thesis [41], and the material libraries from the websites CustomPartNet [51] and MatWeb [52], we have constructed an exemplar hierarchy of device and object attributes. In the attribute taxonomy, attributes are intuitively categorized based on the application field. For example, the properties "Reynold's number" and "Viscosity" are both relevant to devices that work with a fluid medium. Thus, "Material->Fluidic" is the category of the attribute taxonomy that contains these two properties. The full taxonomy can be seen in the NISTIR report [49].

In order to quantify the values of attributes, it is necessary to declare units such as millimeters, dollars, or pounds. Therefore, each attribute may have one or more types of units associated with it. Each unit has an embedded conversion factor for converting to and from SI units, when appropriate. This allows automated tools to perform comparisons in different units for a given attribute [53].

Phrases often will also contain mathematical constraints, either explicitly or using an implicit, inexact term. These constraints may be simple, such as specifying the value of an

attribute for a particular object in the product environment. An example phrase is "a block with mass of 45 kilograms." This can be represented by mapping the word "block" to an item from the taxonomy, and then writing an expression for the relationship between the mass of this object and 45 kilograms. This is symbolically represented in Matlab syntax, which was chosen because it is widely used and familiar to those in the design community. Attributes are expressed as functions, and the objects, which they measure, constitute the arguments of these functions. For example, this construct would be represented as "mass (block) = 45". The attribute is also mapped to a word from the taxonomy (mass), as well as the selected units (kilograms).

Even phrases, which contain terms, which could implicitly suggest a mathematical constraint, can be defined using an explicit mathematical expression. For example, consider a phrase that contains the word "cloudy." When the phrase is defined, an attribute, "irradiance," and a range of values [0, 120] $W/m^2$ can be used to provide a more rigorous definition for the word "cloudy." Both explicitly written phrases such as "irradiance between 0 and 120 $W/m^2$," and implicit phrases such as "cloudy days" can thus both be defined and encoded using a single, common representation format.

More complex expressions can also be represented, such as mathematical formulas, constraint equations such as "$R - R\cos\theta = 6.5$", or direct mappings of values ("voltage across pins=3.5V => connection color=blue"). For author convenience, attributes can be represented as symbols, and these symbols can then be used to build formulas or associative arrays using the Matlab syntax [54]. Once again, symbols are mapped to attributes in the taxonomy, and units are selected.

For requirement statements in which quantities have been explicitly defined, natural language tools may be used to automatically extract and generate mathematical constructs as represented in our framework. For statements in which quantities and values are left as implicit or inexact, our framework provides a prompt for the author to supply more detailed information.

## 3.7 Example of PDS Authored using our Framework

To show the applicability, benefits and limitations of our framework, we will consider the design of an Automatic Pot Stirrer [55] as our case study. This case study is loosely adapted from a Capstone Design Class at the University of Maryland. In this project, using a set of criteria specified by the teacher, the students had to explore different design solutions with the help of engineering tools and techniques. The delivery of the project included a detailed description of the finally adopted design specifications in the form of a PDS document.

The case study is chosen for its simplicity and completeness. Our case study is a representative example of a real enterprise-wide design process; a set of requirements is given as input in the form of a document written in natural English, and a design solution is delivered as output in the form of CAD files and technical documents.

We have already provided taxonomies for the kitchen appliance device category (Figure 3), the corresponding interaction objects (Figure 4), attributes, and verbs.

The requirement statements have been categorized according to their relevance within the product's life and parsed into their constituent phrases. Phrases have been defined, individual words have been mapped using words from the taxonomies, and mathematical relationships have been explicitly expressed.

The full set of requirements is 14 pages long and so is not shown in full in this paper. In Figure 6 we present some example device declarations, interaction object instantiations, and statements that have been represented using our framework. Figure 7 demonstrates the XML representation of a requirement statement.

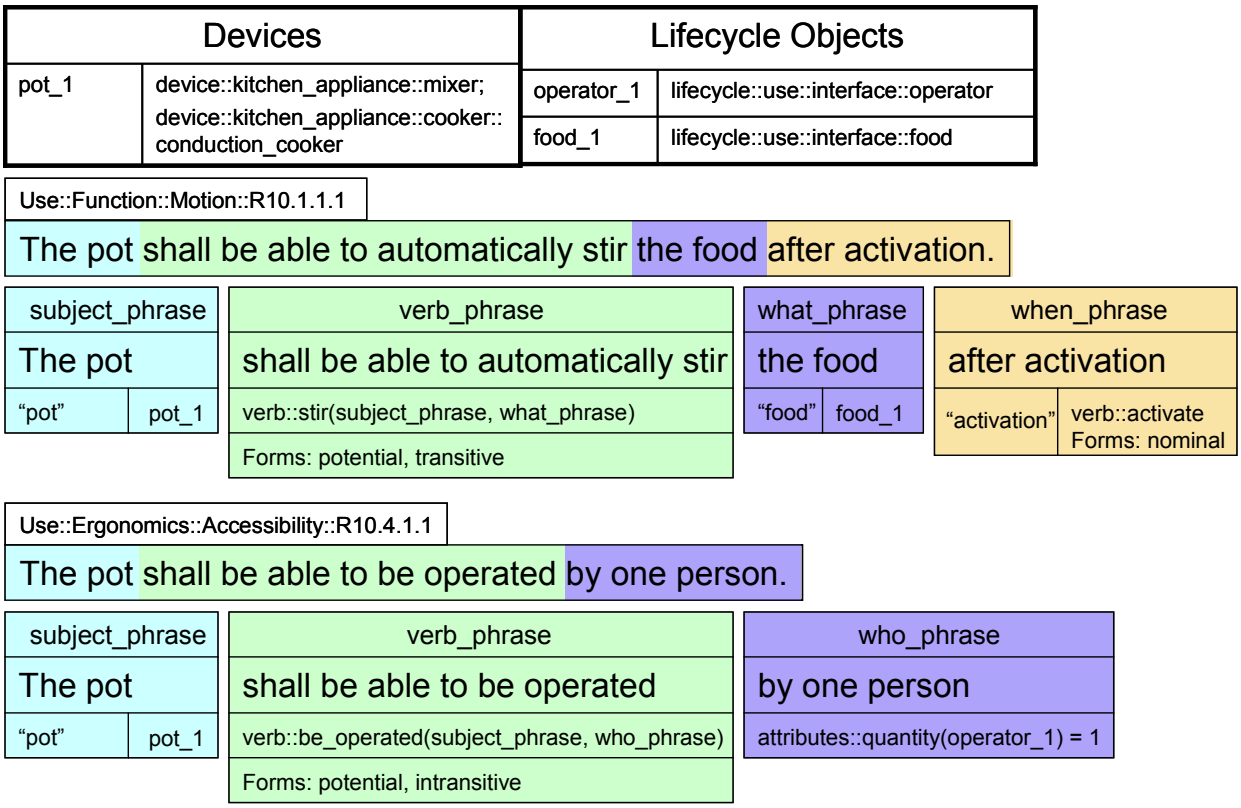


**Figure 6: Sample requirements from PDS for Automatic Pot Stirrer**

```
<requirement_statement id=R1.4.1>
      <label>"The cost of the pot shall be less than $70."</label>
      <subject_phrase id="R1.4.1.SP1">
            <label>"The cost of the pot"</label>
            <object>
                  <label>pot</label>
                  <object_id>pot_1</object_id>
                  <requirement_attribute>
                      <label>cost</label>
                      <link>Attributes\Financial\Cost</link>
                  </requirement_attribute>
              </object>
      </subject_phrase>
      <attribute_mapping id="R1.4.1.A1">
        <object>
            <label>pot</label>
            <object_id>pot_1</object_id>
            <requirement_attribute>
                <label>cost</label>
                <link>Attributes\Financial\Cost</link>
                <symbol>"cost"</symbol>
                <unit>Units\Financial\Dollars</unit>
            </requirement_attribute>
        </object>
        <expression>"cost < 70"</expression>
      </attribute_mapping>
</requirement_statement>
```

**Figure 7: XML representation of a requirement statement.**

The requirement categories proposed in Figure 5 were sufficient to express all of the requirement statements in the PDS. Furthermore, we found that although some statements were relevant to multiple categories, there was always a single, primary category which was much more strongly associated with each statement than any other category. Some categories were left empty, either as "not applicable," such as Public Relations, or "to be determined," such as Installation. This reflects the dynamic nature of the PDS as the design progresses.

Each statement was effectively parsed and defined using taxonomy words and mathematical expressions. Some statements were rewritten to better capture the semantic relationship. For example, "Transmission of power shall be via an electrical cable" was rewritten, as "The power for the pot shall be transmitted through an electrical cable." This was done to make "power" the subject of the statement rather than "transmission," which is better modeled in a verb-phrase. Ultimately, applying an appropriate form to requirement statements is the responsibility of the requirements author.

# 4 Analysis of Results of PDS Case Studies and Usability of our Framework

## 4.1 Overview

To assess the efficacy of our proposed PDS framework, we have taken several examples of real requirements documents and manually encoded each one using defined phrases, statement categories, and lexical taxonomies. The examples we chose were taken from "ENME472: Integrated Product and Process Development," a senior design course offered at the University of Maryland, College Park. We chose this source because it was a convenient source of information and feedback, and because the student authors had little prior experience writing PDS documents. This ensured that our analysis would not be biased by a particular company or organization's *modus operandi* for structuring the PDS. The five devices for which we obtained a PDS were "Sunny Side Up" [56], a solar powered cooking device, "Armstrong" [57], an upper-body bicycle drive system, "Clarity" [58], a rain wiper system for motorcycle helmets, "Third Hand" [59], a vise for irregularly shaped work pieces, and "iSpot" [60], a mechanized spotting system for weightlifting. A hypothetical automatic pot stirrer will be used as running example for this paper. Case studies in section 4.2 will be based on five design projects.

To determine how successful our framework was at satisfying the goals of computer-interpretability and usability, we measured each requirement statement in the five test documents according to semantic mapping and convenience, respectively.

The goal of computer-interpretability was established to ensure that the data represented in our framework could be automatically processed in a reliable manner. Thus, to fulfill this claim, our framework must map as many words as possible to words in the taxonomies. To measure the ease of mapping, we identified the semantically significant elements within each statement. These elements are words or phrases which express a single discrete concept and may be defined or mapped using our framework. For example, the statement "The cooker shall use stored energy with the incident solar thermal radiation to cook food" contains six elements: "The cooker," "shall use," "stored energy", "incident solar thermal radiation", "cook", and "food". We then scored each statement by counting the number of elements that could be appropriately mapped to the taxonomy. For the above example, "The cooker" maps to the device taxonomy, "shall use" maps to the verb taxonomy, "stored energy" can be defined using the verb and object taxonomies, "incident solar thermal radiation" maps to the object taxonomy, "cook" maps to the verb taxonomy, and "food" maps to the object taxonomy. Therefore, this statement would receive mapping success score of 6/6. Scores for each statement are aggregated for the entire document to compute the overall score for the document.

Furthermore, it was necessary to ensure that these mappings would be consistent from user to user. Therefore, we randomly selected 16 well-written requirement statements from across eight different PDS documents generated by senior design course students, and distributed them to four volunteers who were not in the class. The volunteers were given no context about the product for which the statement had been written. Each volunteer was asked to categorize each statement in an appropriate life stage, and then map the words in the statement to appropriate taxonomy words. For each statement, we scored the users based on how consistent their mappings were with each other and with the basic intent of the statement. This was done as a yes/no score, and quantified with a "100%" for correctly mapped statements, and "0" for

incorrectly mapped statements. Mappings to multiple categories or taxonomy words were considered correct if at least one of the mappings was deemed acceptable. The statements that were selected, and the average score, are reported in Section 4.2.

The goal of usability was put into place to ensure that the framework would be easy to apply to current practices, with minimal need to retrain requirement document authors. Largely, our framework automatically fulfills this criterion because it allows authors to write requirements in natural English, rather than in some controlled language. Furthermore, it supports tabular data in the form of attribute/value pairs. Nonetheless, some PDS documents contain information in the form of charts, graphs, and diagrams, which would have to be converted to statements or tables in our framework. Therefore, we measure flexibility in terms of convenience, i.e., how much effort is needed to get a piece of information into a form suitable for our framework. This was measured on the statement level; each PDS was scored based on how many pieces of information would need to be converted to a requirement statement or attribute/value pair. Therefore, a PDS with 51 requirement statements and a diagram that illustrates six semantically significant elements of the design would receive a convenience score of 51/57.

## 4.2 Results

The results from our analysis are presented in Figure 8. For each PDS, we give the aggregate score for mapping success. For these products, all requirements were represented as statements and thus there was no loss in mapping convenience.

| **Product PDS** | **Mapping Success** |
|---|---|
| *Sunny Side Up* | 284/287 (99%) |
| *iSpot* | 263/274 (96%) |
| *Armstrong* | 196/204 (96%) |
| *Clarity* | 251/261 (96%) |
| *Third Hand* | 330/340 (97%) |
| **Aggregate** | 97% |

**Figure 8: Results from analysis of our framework as applied to the five example cases.**

As can be seen, 96-99% of the semantically significant data in the test cases was appropriately mapped using our framework. The taxonomies used for this analysis are unlikely to be exhaustive enough to achieve the same rate of success in industry. To create truly exhaustive taxonomies, the framework must be put into practice and evolved based on feedback from many PDS authors over a sufficient period of time.

For the consistency experiment, we used the following 16 requirements selected from various PDS documents generated by the design course students:

*R1. The grill shall be sold at Walmart.*
*R2. A pressure sensor or an infrared laser that will signal to the motor to start running.*
*R3. The device shall assist in walking up and down stairs.*
*R4. The accuracy of the club placement in concert with the brushes should be very high in order to get a quality clean.*

*R5. The external dimensions shall be 8 x 8 x 6 inches.*
*R6. The overall integrity of the ABS is important to ensure infrequent breakdowns / failures of the system.*
*R7. The product shall appeal to health conscious people.*
*R8. 100% of parts and labor shall be procured from within the USA.*
*R9. The reliability of the switch is critical.*
*R10. The pricing policy over life cycle is TBD.*
*R11. The sensors will be flush mounted.*
*R12. The price will be $25.*
*R13. The walker shall be able to support 300 lbs.*
*R14. The motor shall be powered by batteries.*
*R15. The mechanical work necessary for cleaning shall be performed by a motor.*
*R16. The market size is approximately 78 million homeowners in the United States.*

Correctness scores aggregated for each user varied little from one another, with a mean of 77% and a range from 63% to 81%. Further training would have increased these scores. It was also observed that a large portion of the inconsistencies resulted from interaction object mappings. While the categories, devices, and verbs were consistently mapped, the interaction objects were often either not mapped, or mapped to a very wide range of terms. However, these apparently disparate mappings were still typically related at some level in the taxonomy. This suggests that a more refined interaction object taxonomy may be necessary for practical application.

The studied examples support our claims of usability, customizability, extendibility, and computer-interpretability as outlined in Section 3.1. Usability has been verified through broad range of statements which can be completely defined, mapped, and categorized in accordance with the framework. Our framework also provides for modular customizability: a user can choose whether to use a particular element of the framework at the expense of reduced compatibility. Furthermore, our framework allows the user to scale the system by extending these elements, if the extension does not contradict any existing elements of our framework. This is crucial towards ensuring backwards compatibility of a PDS with the framework. Finally, our framework is computer-interpretable in the sense that every element of every requirement statement can be broken down into semantically typed phrases, and the words can be mapped to one or more words with predefined meanings or to mathematical expressions.

# 5 Software Tool for Categorizing and Labeling Requirement Statements

## 5.1 Overview

In order for our framework to be applied in practice, it is necessary to provide a software tool that accommodates the categorization and mapping schemes presented in Section 3. The scope of this tool is currently centered on authoring PDS documents; future work will build on top of this by creating compatible search and validation tools.

While word processors such as Microsoft Word could certainly be adapted for this purpose through the use of numbered lists and comments, this approach is extremely cumbersome for the user, and risks losing essential data if presented to an author who is

unfamiliar with our framework. Another possibility would be for the author to use an existing XML editing tool to compose and annotate PDS documents which conform to our representation. However without a suitable interface, it would not be possible to parse and define phrases in a semi-automated fashion. The tool presented here uses WordNet and the Stanford Parser [61] to assist the author in identifying and defining phrases, and to select appropriate mapping words from the taxonomies. Integration of these features into traditional XML editing software may not be possible if the software does not support plugins. For this reason, we have built and tested our own software tool.

Our primary goal for this tool was usability; this goal was chosen because the adoption of our framework is crucially dependent on how easily PDS authors can begin applying our framework to their work. Our secondary goal was output quality. To reach this goal, our tool is able not only to store and output data in plain-text XML format, but also to generate traditionally styled PDS documents for inclusion in reports and other technical documents. The output style can be customized through user controls, or by modifying the Extensible Style Sheet (XSL) used to generate the report.

Our framework offers requirement categories that cover the complete life of the product. However, in practice, many life stages will not be relevant to all products. For example, the "Aesthetics" category would likely not be applicable to factory equipment. Therefore, the author may begin by initializing only those categories which are relevant to the particular product. This helps the author to focus on filling in requirements for those categories which are relevant.

Once the relevant categories have been identified, the author may select the relevant interaction objects within each category as described in Section 3.2. Certain predefined, premapped statements associated with those objects can be then be inserted from a database. This feature is yet to be implemented in our software. Building a comprehensive database of requirement statements for various objects is a massive undertaking and shall be left as an exercise for future work.

After selecting the relevant interaction objects within the relevant requirement categories, and the relevant predefined statements, the author may begin writing, parsing, defining, and categorizing additional statements which are specific to the device. Finally, XSL style sheets can be applied to the PDS to produce a clean, formatted document suitable for inclusion in technical reports.

## 5.2 Implementation

We chose Java as our development platform for our software tool, "PDS Author 1.0." We used a standard "Model-View-Controller" approach, as is common in Java applications. Requirements and defined phrases are modeled as XML Node objects. We also automatically parse each requirement statement and store the resulting grammar tree as an XML object. This grammar tree is used to assist the author in declaring, defining and mapping requirement statements. A high level UML diagram of the software is shown in

Figure 9.

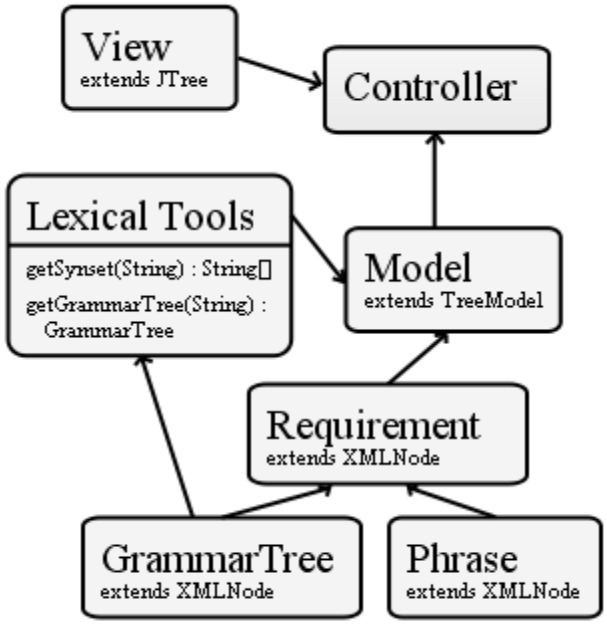


**Figure 9: High-level UML diagram of PDS Author 1.0.**

The master screen for the application (Figure 10) displays the requirement categories and the statements in those categories in a tree layout. When a new PDS is created, he/she needs to specify header data such as author, version, date, and device keyword mappings. Device mappings are then stored for use throughout the PDS. After this initialization step, the new PDS is automatically populated with the categories given in Section 3.4 from a template file.. As new requirement statements are added and defined, the tree layout is updated. Word mappings for the requirements are displayed below the requirement text in less prominent font.

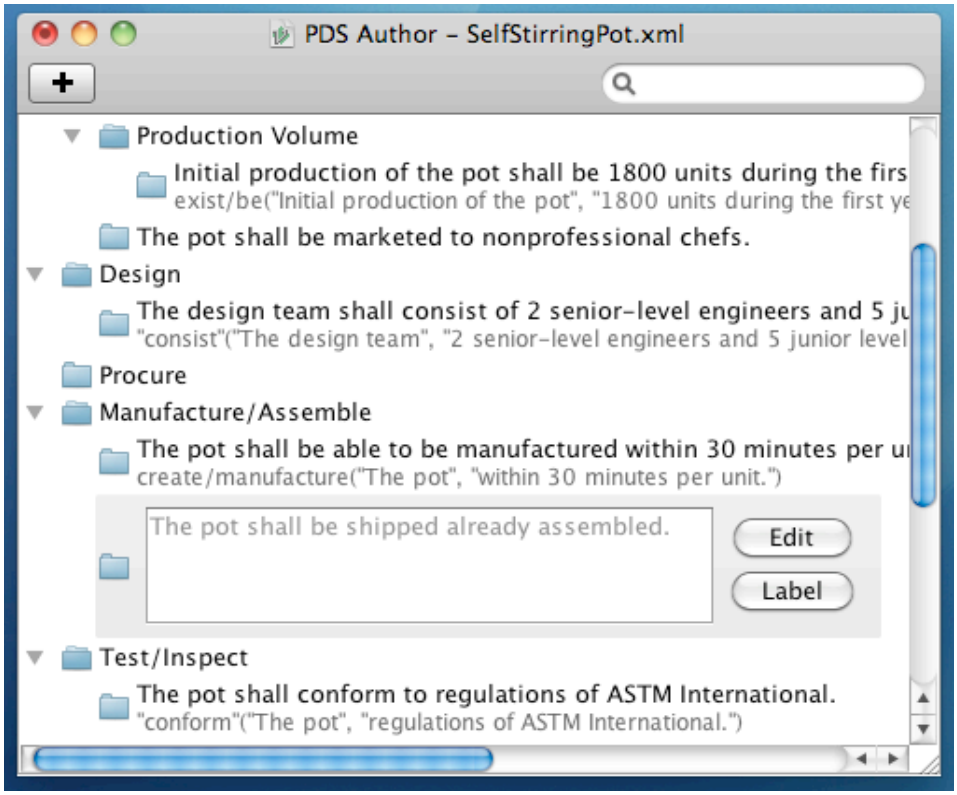


**Figure 10: Master screen for PDS Author 1.0.**

Currently, a new PDS can only be constructed by entering statements directly into the editor dialog, or by copying statements from a Word document and manually pasting them into the editor. The author can then manually define phrases and annotate using taxonomy data. Future work on PDS Author will include a utility for importing requirement statements from plain-text representations such as Word, and then assisting the author by semi-automating the definition and annotation process.

The user can add a new category or requirement to a category by using the main menu or a keyboard shortcut. When a requirement is selected, the user is given the option to edit or annotate it by defining phrases. The edit action makes the statement text mutable, and clears any pre-existing phrase definitions if the statement text is modified. The “define” action opens the phrase definition dialog for the selected requirement (Figure 11), in which phrases can be declared and defined.

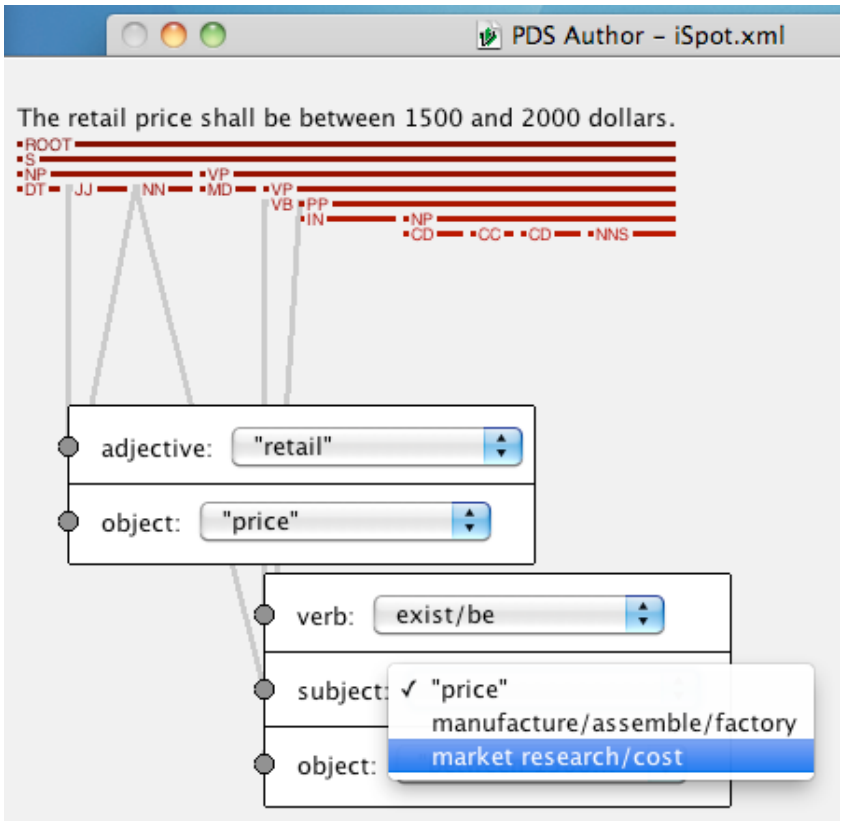


**Figure 11: Phrase definition dialog for PDS Author v1.0.**

Rather than selecting phrases to define directly, they are selected from a phrase structure grammar tree. This grammar tree has the requirement sentence in its root node. A lexical parser then splits this sentence up into phrases, which form the node's children. These phrases are in turn broken up further if possible. Splitting of text into phrases is performed by the parser through analysis of grammatical rules and sentence patterns. Our implementation uses the open-source Stanford Parser to generate a context-free phrase structure grammar representation of the statement. A visualization of this phrase structure grammar can be seen below the requirement statement in Figure 11. This automated lexical parsing alone does not capture the semantic information behind a sentence. It is rather used to tokenize the sentence into smaller pieces which are easier to for the author to map.

The lexical parser also determines the verbs in the statement which might be case frame verbs. The grammar tree streamlines the definition process by allowing users to drag a connection from these case frame verbs to other tokens in the tree, which constitute the arguments of the case frame. Thus, a case frame phrase is completely declared when all its parameters have been connected to items in the grammar tree.

Each action and its parameters have drop-down menus associated with them. From this menu, one may choose an appropriate mapping from taxonomy. The Wordnet database is used to search the taxonomy for synonyms of the word used by the author. Matches are then presented as suggestions for appropriate taxonomy mappings. Defining mathematical relationships is not yet supported.

All data, including templates, taxonomies, header data, requirement statements and categories, and phrase and word mappings, is stored in the XML format. The full specification is too long to be presented here; a representative sample is given in the NISTIR report [49].

Finally, the author may generate an HTML representation of the PDS suitable for inclusion in reports and presentations. The layout is designed in an XSL Extensible Style Sheet, and can be tailored to meet the needs of individual organizations. This is extremely useful since it requires no modifications to PDS Author or the internal XML data representation. The default template displays the title and header data, followed by the requirement statements in a bulleted list. Mappings are listed below each statement.

Preliminary testing and feedback has been performed as a series of focus groups, drawing volunteers from the undergraduate capstone design teams at the University of Maryland. All volunteers were able to use the tool to effectively enter and map requirement statements. This was done with little guidance (approx. 30 minutes of training), suggesting that the format and layout of controls is intuitive. Users found the Wordnet suggestions to be extremely useful in identifying an appropriate mapping for words in requirement statements. However, some additional comments were offered which could be used to improve future versions of the software. For example, an automated means for identifying quantities in a statement that correspond to attributes and entities would be useful. Furthermore, volunteers requested a more complete, generic interaction object taxonomy, which will be developed as a part of future work.

# 6 Application to Retrieving Relevant Requirements for New Designs

## 6.1 Overview

As mentioned in Section 3.1, one motivation for standardizing the representation of requirements in PDS is to allow for efficient searches that go beyond the capabilities of a simple keyword-based search. By searching on the metadata encoded in each statement according to our framework, and the relationships between those pieces of data, we can locate requirement statements pertaining to very specific circumstances. Finding these requirements is valuable for designing new products which may share some of these specific circumstances with previously designed products.

In the design community, solving new design problems by identifying commonalities with previously solved design problems is known as case-based reasoning (CBR). CBR consists of four steps: retrieval of a relevant design case or cases from a knowledge base, adaptation of the retrieved cases to the new problem, evaluation of the new solution (for example, using simulation), and enrichment of the knowledge base with the new design solution [62]. One of the major challenges in CBR methodologies is efficient retrieval of design cases [63]. In practice, these design cases may be indexed by design patterns [64]. By establishing a consistent representation of words and phrases in PDS documents, our framework could alternatively allow design cases to be indexed by their requirements. Designers could then solve new problems by retrieving previously solved problems with similar requirements.

Another useful application of our framework is in determining requirements traceability. As mentioned earlier, models such as SysML [31] and the RDF reference models discussed in

[36] allow users to trace the paths of dependency from design elements back to requirement statements. However, the individual links between requirements and design elements must be established manually, which can be tedious for very large projects. Our framework allows for this process to be partially automated, by comparing the language used in requirement statements with the language used in the product model. If both the requirements and product model are created using a consistent vocabulary such as that presented in our taxonomies, natural language processing may be applied to form many of the links in the traceability network.

Once a traceability network has been established, our framework can also be used to enhance verification procedures. Verification consists of checking that all requirements have been fulfilled by the design solution [65]. Parameters of a design element that are explicitly stated in the design model can be traced back to the relevant requirement statements. These parameters can then be checked against any constraints that appear in the requirement statements. Because our framework allows for constraints to be explicitly defined in a consistent manner, verification can be automatically performed in a reliable fashion. This section presents an example of some complex queries, which can only be performed on PDS documents represented using our framework. These queries would be difficult to perform using documents represented in plain text, because requirement statements can be written in many different styles, using various synonyms. A study conducted by Golden reported significant variations in the style and content of the requirement statements as a part of his study [41]. Four users independently generated requirement statements stating that the CD player should be able to play MP3 files. However, each user's representation was highly unique, as demonstrated in the table in Figure 12.

| **Student** | **Requirement Statement** |
|---|---|
| 1 | Play MP3s |
| 2 | Make CD player compatible with MP3 data files so that MP3 CDs can be played |
| 3 | Can use MP3, CR-R, CD-RW, CD formats |
| 4 | Able to play different formats |

**Figure 12: Sample requirement statements for a CD player.**

Student 2 gave the most detailed requirement, specifying that CDs containing MP3 data files would be played on the CD player, while student 4 gave the least detailed requirement, simply stating that the player should be able to play different formats. Furthermore, some students used the words "play" and "MP3 data file," while others used the less specific word "use" and "format." Our framework helps to alleviate these problems by mapping these imprecise words to more precisely defined keywords. A future version of the authoring tool could help the author to automatically identify and resolve these potential ambiguities and lack of detail.

Performing complex searches on plain text PDS documents using keyword search is time consuming and prone to errors. Although many of the complex queries supported by our search tool could potentially be decomposed into a series of simple keyword searches, studies [66-68] have also shown that finding information using keyword search alone is time-consuming and frequently prone to error. A French study [67] has investigated keyword search efficiency in a group of 32 life science researchers who were given various bibliographic search tasks using the PubMed online medical research database. The study sought to investigate the effects of domain

knowledge (in this case, neuroscience) and resource knowledge (PubMed) on the speed and accuracy of reference searching. The situation with PDS is analogous; the domain is engineering design, and the resource is a database of PDS documents. Ultimately, the study found that even highly specialized neuroscience researchers with good knowledge of their domain did not perform any better than their more generalized counterparts. Timing results from the study showed that the average time that a neuroscientist spent on a search task ranged from 295 seconds to 484 seconds. Results also showed that anywhere between two and four keywords were used in each search query, and that researchers changed their keywords up to 5 times per task. We can therefore conclude that domain-specific knowledge in engineering is also unlikely to significantly improve search speed, and that queries in a similar technical context would need to be changed just as frequently. One common type of keyword change in this study was a change in word form such as "cell" from "cellular," or a change to a synonym of a word such as "ethanol" from "alcohol." Our framework solves this problem by allowing the user to select a single, representative keyword for search, to which all other synonyms and word forms have been mapped.

The study also found that an increased amount of time spent searching correlated with a lower rate of error. Spelling errors, errors in proper search scoping, and errors in properly constructing the query made up over 75% of the errors found. Spelling errors are virtually eliminated in our framework because the user selects search terms from a taxonomy, in which all words have been manually spellchecked by experts. Our framework solves the scoping problem because there is a consistent categorization scheme for requirement statements. Therefore, an author looking to write safety requirements can simply search the safety requirements of other documents. The scope can be easily and reliably expanded by going up one level in the category taxonomy to "Use/Operate" requirements. Finally, accurate queries are easier to construct, because relationships between keywords can be explicitly defined and constraints on quantitative values can be set.

## 6.2 Types of Queries

Suppose a hypothetical company, which designed the Automatic Pot Stirrer as well as hundreds of other kitchen appliances, is now designing the "Easy Loaf," a bread maker for elderly people living independently. Like the Automatic Pot Stirrer, Easy Loaf is a consumer appliance, which stirs food, and applies heat. Unlike the Pot Stirrer, however, this product is specifically marketed to the elderly. Furthermore, based on recent market research, environmentally conscientious consumers demand appliances which can easily be recycled.

If the PDS for the Automatic Pot Stirrer, along with hundreds of other PDS documents, were represented using our framework and stored in a searchable database, we could form specific queries to retrieve requirements relevant to the Easy Loaf. Suppose we are writing the safety requirements for this new device. Since this device will generate heat and be used by elderly people, we might like to know what safety requirements were written for past devices, which both generated heat and were marketed to the elderly. Using our framework, a query could be constructed to search all safety requirements for devices in which the "Function" category and subcategories contain objects with a temperature property above 60 degrees Celsius, and in which the "Accessibility" category contains references to people whose age is greater than 70.

Since plain text can represent the same information in different ways, performing this query on plain text would be very difficult, if not impossible. For example, the temperature

threshold could be represented either as “no less than 140 degrees Fahrenheit,” or as “greater than 60 degrees Celsius.” This is semantically equivalent, yet would not be detected using plain text matching. Furthermore, our framework allows us to search in context; we are only concerned with functional objects, which have this temperature property. A requirement such as “the device shall be able to endure temperatures greater than 60 degrees Celsius” applies to the environmental conditions and is irrelevant with regard to safety.

Other queries could take advantage of the fact that our framework captures relationships between typed phrases. Since the Easy Loaf stirs food (the dough), we might search functional requirements which contain the verb-phrase “to stir” used in its transitive sense, and in which the subject-phrase contains words mapped as stirring devices, and the what-phrase contains words mapped as “food”. We do not know the exact words that the author will use in writing the statement, but we do know which terms the statement will likely be mapped to, and in which types of defined phrases the terms will appear. Therefore, our relational search query can return requirement statements, which match this pattern. Once again, performing this search would be extremely difficult and would require significant natural language processing with plain text.

Finally, because quantities are well modeled we can perform quantitative queries, which seek to find the ideal range for a particular attribute based on multiple intersecting ranges specified by different requirements. Suppose we would like to know the range of acceptable values for the melting point of the material used to create the loaf pan. The Easy Loaf must be recyclable, but will also be subjected to high temperatures. Searching the database for melting point ranges for recyclable metals, we find that materials are easiest to recycle with a melting point below 700 degrees Celsius. However, the database also finds that similar devices used to cook food must be able to withstand temperatures up to 300 degrees Celsius. This query would therefore automatically return a range of [300, 700] degrees Celsius as an acceptable range for the melting point of the material. Once again, a simple text-based query would have trouble finding and intersecting these quantitative values.

## 6.3 Search Precision

To assess the advantage of searching requirements using our representation over a simple keyword-based search, we devised an experimental approach. Eighteen students in a graduate-level engineering design methods course at the University of Maryland were asked to imagine themselves in a particular design scenario and generate up to five queries for searching requirements. A total of 78 queries were generated. Three different scenarios were created, such that six students worked on each scenario. Search queries were generated in plain English. Details of our representation were purposefully withheld to avoid bias, and so students were not asked to add metadata using our representation. Of the 78 queries generated, five examples are listed below.

*Q1. Find all relevant requirements for a device that will heat pots and pans without utilizing an oven.*
*Q2. Find all relevant coatings that do not breakdown at 200C and not stick.*
*Q3. Find all relevant requirements for a system used to manipulate objects in a lack of gravity.*
*Q4. Find all relevant requirements for a mechanical device operated by a remote operator.*
*Q5. Find all relevant requirements for a device that uses explosives in a confined volume.*

For each query generated, we mapped it to our representation and then performed search using both a traditional keyword-based approach, and using the relationships and mappings expressed in our representation. Because mapped words are represented in a hierarchical fashion, searches can be generalized to find results which may use words related to a queried word rather than the word itself. Therefore, for our analysis we introduce the concept of "search depth". We define "search depth" as the number of levels up from the queried object in a taxonomy tree that we traverse until we find an object which, together with its children, form a subtree that may be considered for results. This is illustrated in

Figure **13**.

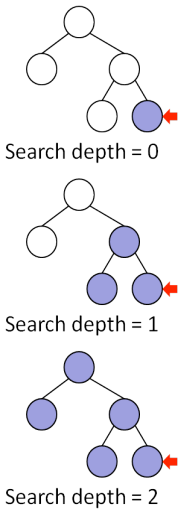


**Figure 13: Definition of "search depth".**

We attempted to match the mapped words in each query to words in a pool of requirement statements. This was done through the various taxonomies, using a specified search depth. Requirement statements that matched all of the mapped words, or words related to the mapped words at the given search depth, were found. These results were then filtered so that only those statements in which the words were used in the same case (Section 3.5) as the query were returned. Results can also be filtered by specifying related taxonomy words to exclude using, for example how-phrases (Figure 14).

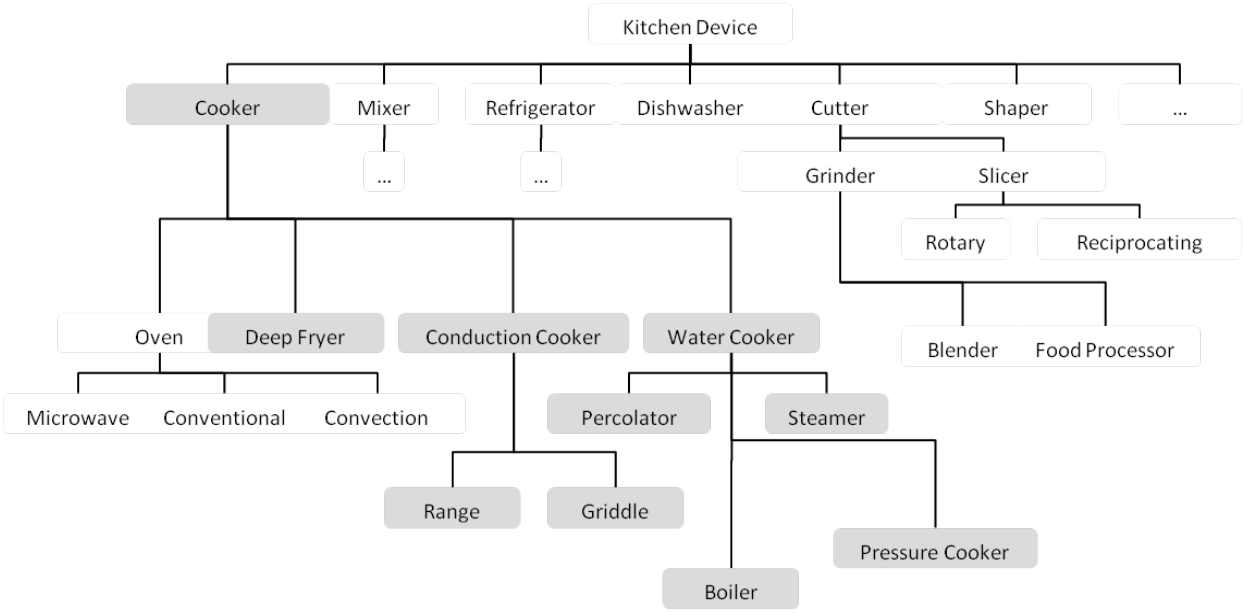


**Figure 14: Searching for cooking devices that are not ovens using the how-phrase "without utilizing an oven".**

Analysis of these queries showed that performing keyword-based search leads to good results in only 5% of the queries. In the remaining cases, either too many or too few results are reported.

Consider the example query Q1. In this case, the search using the keywords "pots and pans", "utilize", "without", and "oven" leads to many irrelevant requirements such as:

*R1. The oven shall heat food without utilizing pots and pans.*
*R2. The oven shall be utilized to heat pots and pans without creating thermal stresses.*
*R3. The pots and pans shall reach 400F in 10 minutes when utilized without opening the oven door.*

In query Q1, search results found using our representation included such statements as "*Food shall be able to reach temperatures ranging from 170-400 degrees F in 10 minutes."* This is relevant to a hotplate because the goal of a cooking device is not only to heat cookware, but the food as well. This statement is tagged with the device "conduction cooker", the verb "cook" and the object "food". Because the statement object "food" matches the query object "cookware" at a search depth of 2, the device is not part of the "oven" subtree, and the device, verb, and object are used in the same case, this is a match. Clearly, this is a reasonable requirement for a consumer hotplate. For the remaining queries, search using our representation was found to return substantially fewer irrelevant queries.

## 6.4 Search Tool

The search wizard allows the user to perform a relational search as described in Section 3.5 and the second example given above. Quantitative searches will be supported in future versions of the tool. Basic keyword searches may still be made by using the search field in the tool bar. Searching is Wordnet assisted, so synonyms of the search term are also matched. Boolean operators are also supported. Taxonomy items may be used within queries. A taxonomy selection panel is used to quickly filter out words by synonyms. Multi-document searches may

be performed on a directory, the results of which are displayed as an HTML document. The user interface for the search wizard is shown in Figure 15.

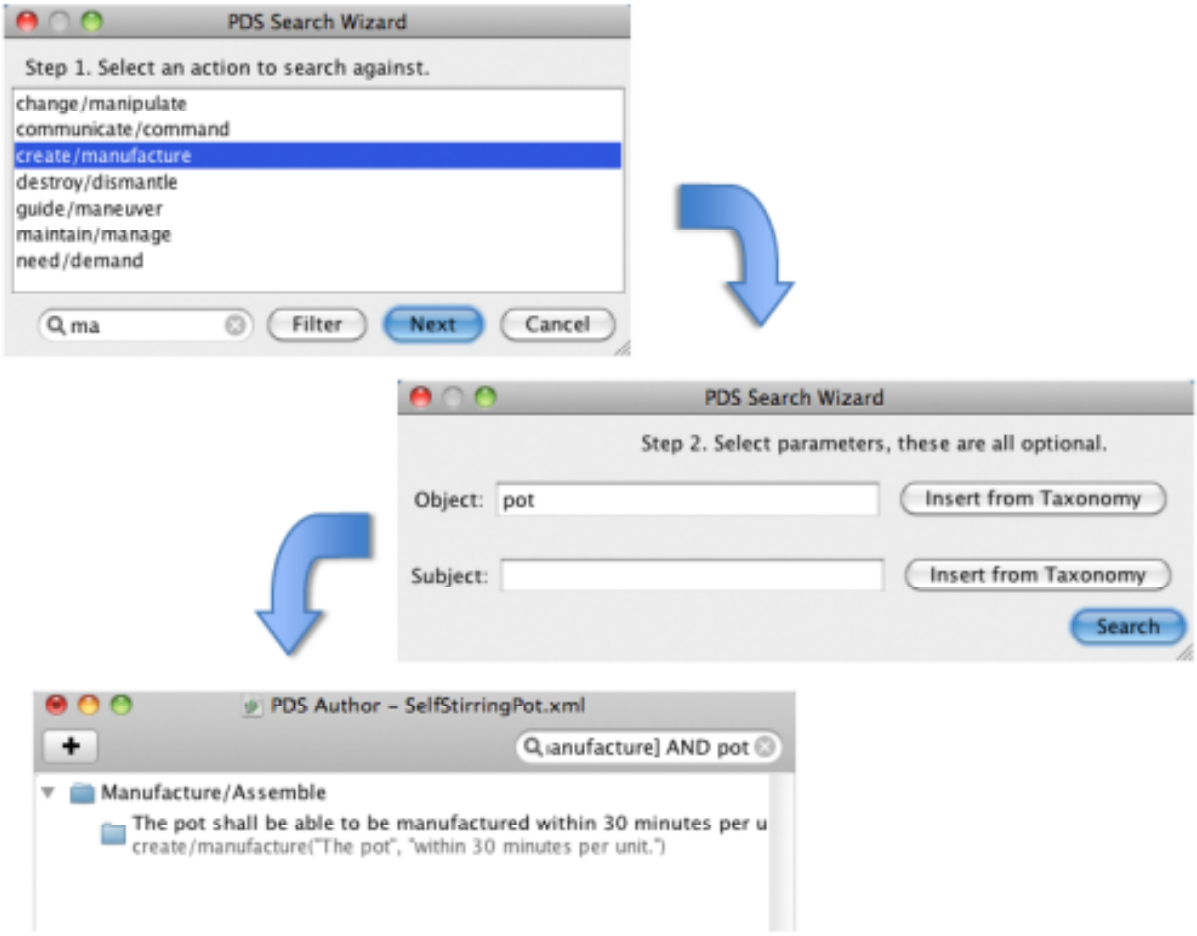


**Figure 15: User interface for performing requirement searches based on phrase relationships.**

# 7 Conclusions and Future Work

PDS authors today wrestle with issues such as writing thorough, meaningful requirements and conveying information in a consistent and irredundant manner. To compose a thorough and usable PDS for a new product from scratch, and to search previous PDS documents for relevant requirements requires significant time and expertise. To enable PDS authors to write PDS in a consistent manner, and to elicit sufficient information from the author on which design decisions can be made, we have developed a framework for authoring PDS. The framework offers a means of defining phrases in requirement statements using quantitative values, a system of taxonomies (verbs, interaction objects, attributes of devices and objects, and their associated units), and other data. Requirements categories are then used to classify the requirement statements based on the product life, and to associate them with interaction objects.

Analyzing representative examples, appropriate device and object taxonomies were developed, and most requirements for the devices were successfully categorized and mapped

with a minimum of effort. This shows that a typical, traditionally written PDS can be encoded into a human-usable, computer-interpretable representation using our framework. Our framework supports the definition of phrases using quantitative values, keyword taxonomies, and other metadata. The taxonomies we present ensure a consistent semantic mapping because they are derived from taxonomies of engineering attributes and verbs that were shown to be effective in ensuring consistency [41]. Further improvements can be made by allowing the taxonomies to evolve to meet the needs of PDS authors.

For this framework to be of practical use in the product development community, a software tool - PDS Author - was developed. Our goal was to develop a tool for efficiently writing and mapping requirement statements, and generating a clean and easy-to-follow output. Based on preliminary testing, it appears to meet both of these needs. If this framework is then inserted into a standard requirements management system such as DOORS, then collaborative PDS authoring can be achieved using the tools provided with these systems. Our framework helps to improve collaborative authoring by ensuring a consistent vocabulary and phrase definition structure.

We have developed a search tool that complements the authoring tool to allow the data that is collected in PDS Author to be applied for requirement reuse. Data from past studies on searching show that the traditional, keyword-based approach is slow and prone to error. This is because a single concept can be represented by many different synonyms. Irrelevant results may also be returned because some requirements might use the queried keyword in a completely different meaning. Using our framework and search tool on the other hand, words with a common meaning are all mapped to the same taxonomy word. Therefore, it is only necessary to search on a single taxonomy word. The use of uniform requirement categories further helps to narrow the scope of the search based on the relevant stages of the product life cycle. The need to consider all possible synonyms of a query keyword, and the frequent return of irrelevant requirements, were the main issues found to contribute to the time and error rate of searches in past studies. By solving these problems, we can conclude that application of our framework will improve search time and accuracy.

Search functionality expands the semantic range of queries. Search can be performed within and across multiple PDS on grammatical, mathematical, and categorical relationships in addition to keyword searches. In the future, we will develop a tool to allow designers to automatically create links between requirement statements and design elements (e.g., functional elements modeled using CPM, part models created using CAD, and assembly models created using OAM), which can be used by a validation engine to check that a product design has met all of its requirements, and identify critical design elements. It is hoped that this will further add to the value of the framework further in the design process, and for subsequent derivative designs. Other future directions for extending this work include providing formal links between PDS and CAD model and PDS and Technical Design Packages. This work can also be used to create foundations for long term archival of engineering designs.

The limitations of our framework are primarily driven by the heavy initial investment required in creating suitable attribute, verb, and object taxonomies. This will require collaboration between many different institutions, which must come to a consensus regarding the words, relationships, and definitions to be used in these taxonomies. The Simplified Technical English dictionary may be a good basis for construction of these taxonomies. Our framework is also limited in that it does not provide a convenient way to represent graphical data, such as

charts and diagrams. However, image formats such as scalable vector graphics (SVG) [69], which are already represented in XML, may be possible to integrate into our framework.

**Disclaimer**

No approval or endorsement of any commercial product by NIST is intended or implied. Certain commercial software are identified in this report to facilitate better understanding. Such identification does not imply recommendations or endorsement by NIST nor does it imply the software identified are necessarily the best available for the purpose.

# 8 References


1. Hull, E., K. Jackson, and J. Dick, *Requirements Engineering*. 2002: Springer.
2. Magrab, E., *Integrated Product and Process Design and Development: The Product Realization Process*. 1997, New York: CRC Press.
3. Dieter, G.E., *Engineering design: a materials and processing approach*. 3 ed. 1999: McGraw-Hill.
4. Safayeni, F., et al., *Requirements engineering in new product development.* Communications of the ACM, 2008. **51**(3): p. 77-82.
5. Chakrabarti, A., S. Morgenstern, and H. Knaab, *Identification and application of requirements and their impact on the design process: a protocol study.* Research in Engineering Design, 2004. **15**(1): p. 22-39.
6. ASD, *ASD Simplified Technical English*, in *ASD-STE100*. 2007.
7. *ASD-STE100 training*. 2009; Available from: http://www.asd-ste100.org/training.htm.
8. Zeng, Y., *Environment-based formulation of design problem.* Transactions of the SDPS: Journal of Integrated Design and Process Science, 2004. **8**(4): p. 45-63.
9. Chen, Z. and Y. Zeng, *Classification of product requirements based on product environment.* Concurrent Engineering Research and Applications: an International Journal, 2006. **14**(3): p. 219-230.
10. Zeng, Y., *Recursive object model (ROM): modeling of linguistic information in engineering design.* Computers in Industry, 2008. **59**(6): p. 612-625.
11. Chen, L. and Y. Zeng. *Automatic Generation Of Uml Diagrams From Product Requirements Described By Natural Language*. in *ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*. 2009.
12. Chen, Z., et al., *Formalisation of product requirements: from natural language descriptions to formal specifications.* International Journal of Manufacturing Research, 2007. **2**(3): p. 362-387.
13. Jiao, M.M.T.J., *A variant approach to product definition by recognizing functional requirement patterns.* Computers & Industrial Engineering, 1997. **33**(3-4): p. 629-633.
14. Gershenson, J. and L. Stauffer, *Assessing the usefulness of a taxonomy of design requirements for manufacturing.* Concurrent Engineering Research and Applications: an International Journal, 1999. **7**: p. 147-158.
15. Gershenson, J. and L. Stauffer, *A taxonomy for design requirements from corporate customers.* Research in Engineering Design. Theory, Applications, and Concurrent Engineering, 1999. **11**: p. 103-115.
16. Darlington, M.J. and S.J. Culley, *A model of factors influencing the design requirement.* Design Studies, 2004. **25**(4): p. 329-350.

17. Wolkl, S. and K. Shea. *A Computational Product Model for Conceptual Design using SysML*. in *ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*. 2009.

18. Aurum, A. and C. Wohlin, *Engineering and managing software requirements*. 2005: Springer.

19. *Guide to the Software Engineering Body of Knowledge*. 2009; Available from: http://www.swebok.org/.

20. Greenspan, S., J. Mylopoulos, and A. Borgida, *On formal requirements modeling languages: RML revisited*, in *Proceedings of the 16th international conference on Software engineering*. 1994. p. 135-147.

21. Greenspan, S., *Requirements modeling: a knowledge representation approach to software requirements definition*. 1984, University of Toronto.

22. Nuseibeh, B. and S. Easterbrook. *Requirements engineering: a roadmap*. in *Proceedings of the Conference on The Future of Software Engineering*. 2000. Limerick, Ireland: ACM Press.

23. Fantechi, A., et al., *Assisting requirement formalization by means of natural language translation.* Formal Methods in System Design, 1994. **4**: p. 243-263.

24. Gnesi, S., et al., *An Automatic Tool for the Analysis of Natural Language Requirements.* International Journal of Computer Systems Science and Engineering, 2005. **20**(1).

25. Almefelt, L., et al., *Requirements management in practice: findings from an empirical study in the automotive industry.* Research in Engineering Design, 2006. **17**(3): p. 113-134.

26. *Telelogic DOORS*. 2006; Available from: http://www.telelogic.com/corp/products/doors/doors/index.cfm.

27. McKay, A., A.D. Pennington, and J. Baxter, *Requirements management: a representation scheme for product specifications.* Computer-Aided Design, 2001. **33**(7): p. 511-520.

28. GEIA, *Processes for Engineering a System*, in *EIA-632*. 1999.

29. IEEE, *Systems Engineering - System Life Cycle Processes*, in *IEEE-15288*. 2004.

30. Sheard, S.A., *Evolution of the frameworks quagmire.* Computer, 2001. **34**(7): p. 96-98.

31. *OMG SysML v1.1 Specification*. 2008.

32. Fenves, S., et al., *CPM2: A Revised Core Product Model for Representing Design Information*. 2004, National Institute of Standards and Technology: Gaithersburg, MD 20899, USA.

33. Fiorentini, X., et al., *Towards an ontology for open assembly model*, in *International Conference on Product Lifecycle Management*. 2007. p. 445-456.

34. *CORE Software*. [cited 2010; Available from: http://www.vitechcorp.com/products/index.html.

35. *Systems Engineering and Requirements Management: PLM−Product Lifecycle Management: Siemens PLM Software*. [cited 2010; Available from:

http://www.plm.automation.siemens.com/en_us/products/teamcenter/solutions_by_product/systems_engineering.shtml.

36. Selberg, S.A. and M. Austin, *Requirements Engineering and the Semantic Web*, in *Institute for Systems Research Technical Reports*. 2003, University of Maryland: College Park, Maryland.

37. Estefan, J.A., *Survey of Model-Based Systems Engineering (MBSE) Methodologies*, in *INCOSE MBSE Initiative*. 2008, Jet Propulsion Laboratory, California Institute of Technology: Pasadena, California.

38. W3C. *RDF - Semantic Web Standards*. 2004; Available from: http://www.w3.org/RDF/.

39. W3C. *OWL 2 Web Ontology Language Document Overview*. 2009; Available from: http://www.w3.org/TR/owl2-overview/.

40. Pahl, G. and W. Beitz, *Engineering Design*. 1984, London: Design Council.

41. Golden, I., *Function Based Archival And Retrieval: Developing A Repository Of Biologically Inspired Product Concepts*, in *Department of Mechanical Engineering*. 2005, University of Maryland: College Park.

42. STEP, *AP 239: Product Life Cycle Support*, in *AP-239*. 2005.

43. *PLIB (http://www.plib.ensma.fr/)*. 2009.

44. ISO, *Industrial automation systems and integration -- Parts library -- Part 1: Overview and fundamental principles*, in *ISO 13584-1*. 2001: Geneva, Switzerland.

45. Thanh, H., et al. *Decision support at the wheel-rail interface: the development of system functional requirements*. in *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*. 2008: Professional Engineering Publishing.

46. Fillmore, C.J., *"The Case for Case"*, in *Universals in Linguistic Theory*, Bach and Harms, Editors. 1968, Holt, Rinehart, and Winston: New York. p. 1-88.

47. Carbonell, J. and P.J. Hayes, *Robust parsing strategies using multiple construction-specific strategies*, in *Natural language parsing systems*. July-December 1983, Springer-Verlag: London, UK.

48. Miller, G.A. *Wordnet: A lexical database for English*. 2010; Available from: http://wordnet.princeton.edu/.

49. Weissman, A., et al., *Formal Representation of Product Design Specifications for Validating Product Designs*. 2009, National Institute of Standards and Technology: Gaithersburg, MD 20899, USA.

50. Ducq, Y., D. Chen, and B. Vallespir, *Interoperability in enterprise modelling: requirements and roadmap.* Advanced Engineering Informatics, 2004. **18**(4): p. 193-203.

51. *Custompart.net: Free Online Manufacturing Cost Estimation and Education Resource*. 2008; Available from: http://www.custompartnet.com.

52. *Matweb*. 2008; Available from: http://www.matweb.com.

53. *Units Markup Language project*. 2009; Available from: http://unitsml.nist.gov/.

54. *Matlab*. 2009; Available from: www.mathworks.com.

55. Chambers, M., et al., *The chef's helper*, in *Final project of the capstone design class*. 2006, University of Maryland: College Park.

56. Hannam, P., et al., *Team Sunny Side Up: Hybrid Solar Cooker*, in *Final project of the capstone design class*. 2009, University of Maryland: College Park.

57. Bergamini, T., et al., *Team Arm-Strong: Multi-Drive Bicycle*, in *Final project of the capstone design class*. 2009, University of Maryland: College Park.

58. Doehner, T., et al., *Team Clarity: All-weather Motorcycle Helmet Attachment*, in *Final project of the capstone design class*. 2009, University of Maryland: College Park.

59. Beard, A., et al., *The Third Hand*, in *Final project of the capstone design class*. 2009, University of Maryland: College Park.

60. Castro, J., et al., *iSpot*, in *Final project of the capstone design class*. 2009, University of Maryland: College Park.

61. Klein, D. and C.D. Manning. *The Stanford Parser: A statistical parser*. 2003; Available from: http://nlp.stanford.edu/software/lex-parser.shtml.

62. Aamodt, A. and E. Plaza, *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches.* Artificial Intelligence Communications, 1994. **7**(1): p. 39-52.

63. Maher, M.L. and A. Gomez de Silva Garza, *Case-based Reasoning in Design.* IEEE Expert, 1997. **12**(2): p. 34-41.

64. Goel, A. and S. Bhatta, *Use of design patterns in analogy-based design.* Advanced Engineering Informatics, 2004. **18**(2): p. 85-94.

65. Duran, A., et al. *Supporting requirements verification using XSLT*. in *Proceedings of the IEEE Joint International Conference on Requirements Engineering*. 2002. University of Essen, Germany.

66. Aula, A. and K. Nordhausen, *Modeling successful performance in Web searching.* Journal of the American Society for Information Science and Technology, 2006. **57**(12): p. 1678-1693.

67. Vibert, N., et al., *Effects of Domain Knowledge on Reference Search With the PubMed Database: An Experimental Study.* Journal of the American Society for Information Science and Technology, 2009. **60**(7): p. 1423-47.

68. Spink A, W.T., Ford N, Foster A, Ellis D, *Information seeking and mediated searching study. Part 3. Successive searching.* Journal of the American Society for Information Science and Technology, 2002. **53**(9).

69. W3C. *Scalable Vector Graphics (SVG) Full 1.2 Specification*. 2005; Available from: http://www.w3.org/TR/SVG12/.