College of Computing Technical Report

CHARACTERIZING THE CONCEPTUAL COHERENCE OF COMPUTING APPLICATIONS USING ONTOLOGICAL EXCAVATION

by

Idris Hsi

Georgia Institute of Technology
May 2004

Abstract

We are investigating metrics for measuring the usefulness of computing applications relative to a specific use context. We define "usefulness" as "the extent to which an application's features succeed in assisting a set of users to achieve a set of goals, relative to the amount of effort required to engage those features." We define a *feature* as a user-accessible behavior or service implemented by a computing application. Computing applications embody and operationalize a set of concepts that correlate to concepts in the domain of the user. The degree to which the application's concepts and the user's concepts agree is its conceptual fitness. We believe that applications with high conceptual fitness to a particular use context will also be perceived as useful by the users in this context. We have chosen in this work to study the problem of measuring conceptual fitness from the application side using conceptual coherence as our unit of interest.

Conceptual coherence is an attribute of conceptual integrity, described by Fred Brooks as the property of a system designed under a unified and coordinated set of design ideas. It is the property of a computing application that measures the degree to which that application's concepts are tightly related. In previous work, we established that applications have core concepts – concepts that are essential to defining that application's features. An application will have a low conceptual coherence if it possesses a disproportionate number of non-core (peripheral) concepts. We intend to show that *the conceptual coherence of an application determines its perceived usefulness to its users, and features with only tangential relationships to an application are less likely to be used and reduce that application's conceptual coherence.*

The set of concepts and relationships contained in an application can be said to be its ontology. We have developed methods for the black-box reverse engineering (excavation) of a computing application's ontology from the user interface and use techniques from the user interface and use techniques from graph theory to identify the core concepts of an application and its teleons – tightly connected functional subgroups within the ontology. We have also developed a technique called use case silhouetting which measures the ontological coverage, the number of concepts activated by a use case or set of use cases, and the relative importance of a concept to a set of use cases as a first approximation of conceptual fitness. We have applied these techniques to four small applications: the Windows 95/98 CD Player, the Palm Pilot Scheduler, Microsoft Notepad, and the Protocol Calculator / Calendar.

We propose to perform two exploratory studies and one confirmatory one. Our first exploratory study will excavate and analyze the ontologies from three large systems – Microsoft Powerpoint 2000, Microsoft Word 2000, and Yahoo Instant Messenger 5.5. Our second study will obtain use cases from an independently written instruction manual (the "for Dummies" series). We will use these to develop use case silhouettes on our excavated obtained from Dr. Joanna McGrenere in her study on adaptable interfaces. We will show that the user preferences expressed by her subjects correlate to data obtained from our ontological analysis and use case silhouettes of Word.

Table of Contents

1 Introduction

1.1 Conceptual Integrity

In his book, *The Mythical-Man Month*, Fred Brooks described a desirable quality of software that he called *conceptual integrity*. This property arises from a system that demonstrates design qualities that could only have been engineered under a unified vision of that system.

> *"I will contend that conceptual integrity is the most important consideration in system design. It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas." [37]*

Brooks describes how conceptual integrity can be seen in the design of a computing application's architecture, user interface, and functionality. He used the example of a cathedral at Reims in France as an example of a structure with such conceptual integrity that it invokes joy in the beholder. In architecture, Alexander observed this sense of coherence, aesthetic, and integrity in smaller integrated units of landscape and architectural features that he cataloged into what he called a pattern language [3, 4]. In computer science, design attributes that suggest similar notions of conceptual integrity have been qualified in items like code [141, 142] and the "bad smells" judgments used to identify awkward code [70], design patterns in software architectures [72], open source systems such as Linux [179], user interfaces [37, 60, 162], and web pages [199]. In architecture, conceptual integrity measures the extent to which the designer unified the building's purpose or concept with the constraints of structure and material. Buildings and cities are designed for mostly one basic purpose: so that their inhabitants can live and work in them. Computing applications have many different purposes but they still require a good design to perform their functions well. But unlike buildings or cities, which have physical constraints that guide their form and composition, the conceptual integrity of computing applications can be much harder to perceive, design, or engineer.

1.2 Problem Domains and Software Ontologies

Applications are engineered to solve problems in specific *use contexts*. A use context consists of the external physical (or virtual) environment that contains the computing application and its users, the goals that the combined computing application/user system wishes to achieve, and the various nuances (business rules, customer demand, user and system capabilities) that govern the operation and performance of both environment and goal completion. For example, the use context of a bank customer database consists of the bank itself, the systems that manage and store the database, the employees charged with maintaining the stored information, and the rules and procedures established by bank management for storing and distributing the data. All use contexts exist within a *problem domain*. Arango and Prieto-Díaz state that a problem domain is a collection of items of real-world information that has "deep or comprehensive relationships among the items of information" and a community that has a stake in solving those problems [11]. Software that has been designed to function in the use context and the problem domain will possess a set of concepts and relationships that we call an *ontology* [35, 67, 81, 147, 203, 204].

The ontology of a computing application can be said to be its theory of the real world. The concepts it embodies determine and structure its *features*, which we define as the *user-accessible*

behaviors and services implemented by the system. Now, Brooks argues that software must make a computer easy to use.

> *"Because ease of use is the purpose, this ratio of function to conceptual complexity is the ultimate test of system design. Neither function alone nor simplicity alone defines a good design." [37]*

Ultimately, users evaluate software on its ability to help them to achieve their goals, whether these goals are entertainment, productivity, scientific, or industrial. Thus, software engineers should be primarily interested in ensuring that their creations have a high level of *usefulness*.

1.3 Usefulness

We define usefulness as *the extent to which an application succeeds in assisting a set of users to achieve a set of goals, relative to the amount of effort required to engage those features.* We distinguish usefulness from *usability* which is an integral but subordinate attribute of usefulness. A useful application with poor usability can still enable users to achieve their goals albeit with great difficulty. An application with little or no usefulness can be extremely usable but cannot help the users to achieve their goals. Developing useful software requires that developers understand what their users are trying to do in a specific use context and encode that knowledge into the design. Yet an application must possess enough features to be useful to its users but without becoming too complex: a design tradeoff between functional power and conciseness. The features are accessed by the users of the system through its user interface or *morphology,* which is the external presentation of the software. Thus what the software is, how it is presented to the users, and how it functions must ultimately be determined by its ontology. If its ontology does not match the user's understanding of the problem domain then the application will fail. If the ontology has been modeled incorrectly, relative to the problem domain, then the most advanced techniques in program design, development, and testing will not be able to produce a useful computing application. In other words, *the usefulness of a computing application is determined by the conceptual fitness of its ontology to the domain of the user.*

A method for measuring the conceptual fitness of an application's ontology to the domain of a use context would allow us to measure the actual and potential usefulness of an application, possibly prior to development. However, measuring conceptual fitness requires both a comprehensive model of the application's ontology as well as an equivalent and comparable model of the user's domain. For this research, we chose to address only the problem of modeling the application's ontology and to leave the full solution of measuring conceptual fitness to future work.

1.4 The Research Problem

We have stated the following:

- Conceptual integrity is a desirable quality in computing applications and is evidenced by a well-designed software architecture, user interface, and feature set.
- Computing applications are developed to be useful to their users and to behave in specific problem domains.
- These applications encode the user's domain in a set of concepts and relationships that we call an ontology.

- The concepts in the ontology determine what features the software implements.
- The degree to which the ontology matches the problem domain of its users is its conceptual fitness.
- An application possessing high conceptual fitness is more likely to be useful than one with a low conceptual fitness.
- The morphology, architecture, and code must necessarily implement the features articulated by the ontology.
- Thus, the ontology is the single most important factor in the conceptual integrity of the application.

The quality of conceptual integrity encompasses much more than the application's ontology and includes other aspects of the application such as its architecture, user interface, and functionality. However, as we argue that these other aspects must necessarily derive from the ontology, we must ask ourselves what an ontology with conceptual integrity exhibits in its design, structure, or composition. The answer can be derived from the idea of conceptual fitness.

In thinking about problem domains such as meeting scheduling, banking, or telephony, we can infer that each of these have a set of concepts that define them. Meetings require participants, scheduling procedures, and a reason to meet. Banking involves financial transactions, customers, and fees. Telephony offers communication services through specific media to connect people to each other. However, some domain descriptions for specific use contexts will include seemingly optional concepts. For example, an alarm that reminds the user of an impending meeting might be helpful but may or may not belong to the defining set of concepts that articulate a meeting scheduling domain. Banks may offer investment advice to their customers, something that may or may not be a central concept in banking. Telephony services can include features like vanity numbers or opinion poll numbers [210], which go beyond basic call connections. These kinds of concepts may be one step removed from a defining set of concepts. If meeting scheduling incorporated types of meetings such as birthdays or anniversaries, generalizing it to events, is it still a meeting scheduler, in the strict sense of a business meeting? If banking services began to include advice about real estate and home ownership or insurance, is it still strictly banking? Some cell phones now include digital cameras in their feature sets that allow you to take pictures and send them to a recipient. Is this still telephony? Since technology evolves with the requirements of its users, these services or the systems implementing these services are probably still exhibiting a high conceptual fitness by maintaining a correspondence between its ontology and the user's problem domain. However, an ontology that has too many concepts one or two steps removed from its essential concepts may have lost conceptual integrity. Such an ontology no longer expresses a single, unified idea but multiple ideas that do not necessarily relate to one another.

We define a quality of the ontology that contributes to the application's overall conceptual integrity that we call *conceptual coherence*. Conceptual coherence measures the degree to which a computing application's concepts are tightly related. *We believe that applications with low conceptual coherence will be perceived as possessing less usefulness than their potential suggests and that features with only tangential relationships to those features essential to an application reduce that application's conceptual coherence.* This intuition was derived from our observations of many commercial systems where each new version adds features and the

complaints from users and reviewers about how these additions hindered their use or enjoyment of the application. We decided to began our investigation by studying software evolution to determine how an application's features changed over time.

2 Studying the Feature Evolution of Software

2.1 Software Evolution and Feature Aggregation

Software evolution refers to the process of growth and change over the lifetime of the software during its maintenance phase. Perry characterizes these changes into three categories [166]:

- corrections – repairs to errors in the code
- improvements – optimizations to performance, usability, maintainability, and so on.
- enhancements – additions of new features, generally visible to the users of the system.

Software tends to go through many iterations of development and enhancement, evolving over time as dictated by the competitive demands of the marketplace and in accordance with Lehman's Law of Software Evolution that states that a computing application (specifically what he calls an E-type program) "must be continually adapted else it becomes progressively less satisfactory" [128]. Ultimately, software must satisfy its users whether its role is to entertain, to facilitate intellectual activities, or to produce work products. Because these goals are embedded in real world contexts, software engineers must contend with two issues in the development and evolution of these systems.

First, specifying and designing such systems so that the embedded domain model has sufficient fidelity to operationalize the services desired by the customers and users of the system is an immensely complicated process for requirements developers [6, 20, 53, 100, 105, 134, 172, 176]. Second, the real world also changes over time. Organizational goals change as do procedures and processes. Introducing new technologies also perturb the original domain as users learn and adapt their own behaviors to these new tools [104, 180]. These changes cause the software's model of the world to fall out of step with the actual world [129]. The first two categories of evolutionary changes, corrections and enhancements, simply improve an application's ability to implement its current domain model. To change that model requires that it be enhanced. Usually, this enhancement is accomplished by adding features to the application [38, 130, 143, 207]. We call this process *feature aggregation* although it is also called, more critically, *feature creep* and *creeping featurism* [162].

Does software actually become more useful over time? Do these enhancements improve its usefulness to its users? An engineering expectation might be that they do. Artifacts such as forks, pencils, paper clips, and bookcases are adapted over time until they have a stable set of optimized features that allow them to perform their function well [167, 169-171]. Forks develop extra tines, pencils acquired a wood casing around their lead interiors, paper clips changed in shape and length, and bookcases develop movable shelves. Large and seemingly immutable structures such as buildings are adapted and improved over time to meet the needs of their inhabitants [36]. Even structures that are not inherently adaptable, like bridges, will see new design improvements with each new construction as technology improves and engineers learn from the failures of past efforts [168]. Thus, in engineering disciplines, development techniques improve over time and later generations have better designs and more functional stability than the earlier ones. Software is adapted to optimize its functions but it also adds more features as it evolves – something that is difficult to do to a physical construct. Adding features allows each successive version to perform more functions and gives its users more services. From a

consumer's point of view, one could argue that given the choice between two equivalently priced versions of software, the one with more features will be more attractive because of its potential usefulness. What is unclear is whether this type of evolution, driven by a combination of industrial, marketing, and consumer pressures, has truly made computing applications more useful to their users.

There have been studies to suggest that this form of software evolution does not necessarily produce a new version with increased fitness. A seminal study conducted by Lehman and Belady on the IBM OS360 showed that as that the system aged, it becomes less stable over time [129]. This decrease in stability made the software more difficult to maintain as its code became more complex. Lehman's Laws of Software Evolution argue that programs must continue to grow in functionality to maintain user satisfaction. At later stages in their evolution, they become more difficult to enhance because of their growing complexity [128, 129]. Thus, from a software engineering perspective, adding features becomes a two-edged sword in that features have to be added but they add to the complexity of the system making it increasingly more difficult to adapt and improve.

From the user's perspective, a certain point in an application's evolution, as has been noted in both the academic and popular literature, its user population begins to complain about the difficulty they have with the latest version [15, 74, 144, 146]. These difficulties include applications having too many features, automated features that are not desired, and problems with navigating the user interfaces to find the desired features [37, 126, 157, 161, 162]. Users describe such systems as *bloated*. We formally define *bloat* as *the description applied to applications when it possesses a disproportionate number of unnecessary features that interfere with normal or desired interactions with the application*. The application has lost conceptual fitness by embodying more concepts than are desired by its users.

Evolving computing applications by adding features can also result in difficulties for developers. Researchers in telephony have identified what they call the *feature interaction problem* where proposed features contradict or interfere with existing features [34, 43, 210]. This reflects a tension between the changing services desired by the customer and the established ontology of the software as encoded by developers. Techniques are being devised to accommodate or reduce the introduction of features which conflict with existing functionality [42, 124]. Nevertheless, this problem seems to hint at the idea that application ontologies may have actual limits to their growth at least as far as usefulness is concerned.

It would seem that from these studies that computing applications do not become more useful over time, or that improvements to their usefulness have different costs and implications than one might expect from then engineering or architecting of a physical construct. We may conclude that evolving programs in such a manner implies an entropic process as the system decreases in stability over its lifetime [182]. Nevertheless, by studying software ontologies, we may learn what makes them prone to entropy and decreased stability. Specifically, if software is becoming less stable as features are added to them, we may be able to detect this process by studying the *diachronic variation* (variation over time) of its concepts and to examine applications that have been through several generations to see whether they exhibit increased complexity and decreased coherence in their ontologies.

2.2 The Feature Evolution of Microsoft Word

To study the ideas of ontology and the evolution of an application's conceptual coherence, we began with the following questions:

- How do computing applications evolve their features over time?
- How does the evolution of a computing application affect its perceived usefulness?

Work in design evolution by Henri Petroski show that tools evolve and improve over time over many iterations through combinations of design failure, optimization, and cultural co-evolution [168, 169]. We could make the general claim that "all tools improve with each successful version." However, software lacks the physical constraints and single-minded design of artifacts studied by Petroski. Thus, we needed to study the service evolution of a computing application to learn what happens. We analyzed three versions of Microsoft Word (MS Word 2.0, MS Word 95, MS Word 97) using features as our unit of analysis [98]. In this study, we treated features as units of software function or usefulness. Using the user-accessible elements of the applications, we tracked the evolution of their feature architectures. We used the following tripartite view in our analysis:

- The *morphological view* is the user-visible analog for feature architecture of the source code content of a software architecture. It consists of the user-interface composition and navigation structure.
- The *functional view* is the description of what the features do. A thorough analysis of functionality would require a detailed model of interactions based on data flow or control abstractions. In this research, we restrict ourselves to enumerating *operations*, the activities that the system performs.
- The *object view* is a description of the subject-matter of the feature. Like an object model produced during software design or an information model for database design, the object view consists of static relationships between objects in the problem domain. In the case of the feature architecture, however, the objects are derived from user-visible phenomena, especially the user interface components from the morphological view. The objects in the feature architecture may be correlated with the objects underlying the implementation if it is object-oriented or the data structures and files if it is not, but they need not be. Again, it is the problem domain that makes the products' objects appropriate or inappropriate, not the fact that they are to be recovered from the code.

We also coined the term *teleon* to label a cluster of related concepts systematically derived from our object view. Using black-box reverse engineering methods, we treated elements of the morphology as portals to the underlying operations and objects and built representations for each view for each version of MS Word.

In our analysis, we discovered the following:

- Word's morphology increased in depth and complexity over time. However, these changes were driven primarily by changes and enhancements to the objects. The number of operations also increased over time. While this also correlated to the number of objects added to each version, there were no discernable patterns to how this occurred.

- Objects from previous versions remained the same despite changes to the overall object view. Features did not disappear from a new version with one or two exceptions (in which they migrated outside the application). Thus, older features become more entrenched over time and develop more operations and morphological elements that activate them.
- Teleons were added in "clumps" to the periphery of the object view. Rather than an even pattern of growth where teleons acquired concepts and grew larger, like the annular rings of a tree, new teleons with new concepts and operations were added to the previous set of features. The list of teleons identified in Word 2.0, Word 95, and Word 97 can be found in Table 1.

Table 1 – Conceptual Evolution of the Document Teleons in MS Word

Word 2.0 Teleons	Word 95 – New Teleons	Word 97 - New Teleons
Annotation	**Caption**	**3D Direction**
Border	**Cross-Reference**	**3D Lighting**
Character	**Database**	**3D Object**
Column	**Drawing**	**3D Surface**
Document	**Drawing Object**	**Comment**
Envelope	**Font**	**Font Animation**
Field	**Font Effects**	**HTML Document**
Font Style	**Font Underline**	**OCX Object**
Footer	**Form Field**	
Footnote	**Heading**	
Frame	**List**	
Header	**Note**	
Index	**Numbering**	
Line Numbering	**Revisions**	
Object	**Table of Authorities**	
Page	**Table of Figures**	
Paragraph		
Picture		
Section		
Shading		
Style		
Summary Info		
Tab Alignment		
Table		
Table Cell		
Table of Contents		
Tabs		
Word		

From these findings, we arrived at the following conclusions:

- MS Word evolved most noticeably by adding new features to the previous version's set.
- The user interface or morphology is an inadequate point of analysis for understanding an application's complexity or usefulness. Morphologies are driven by the application's underlying theory. Morphological complexity can affect ease of access or activation of certain operations, but the application's overall usefulness is determined by the concepts it contains and implements.
- Older features may remain because they define the application or to preserve compatibility with other applications. In MS Word, if a fundamental word processing concept such as "word" or "paragraph" were to disappear in the next version, then it would no longer be a word processor.
- 'Bloat' results when adding newer features interferes with access to older features.

Finding that older features persisted in Word suggested that all applications, new or evolved, have an ontological foundation composed of concepts necessary for the definition of those applications. We also believed that if applications do consist of organized clusters of concepts that compose features that these can be identified from their ontologies.

3 Ontological Excavation

We developed a technique called *ontological excavation* to reverse-engineer the ontology from the morphology [99]. Ontological excavation uses black-box techniques; the ontology is reverse engineered from the user interface of the application rather than the source code. Black box reverse engineering allows us to identify just the concepts visible to the user rather than the concepts relevant to the application's implementation. The basic steps of ontological excavation are:

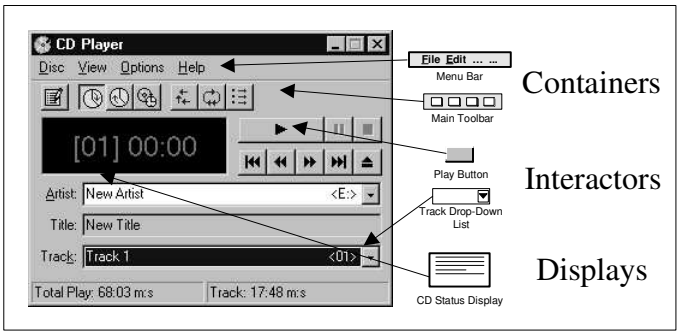
1. Model the user interface in a *morphological map* of the application's interactors, displays, and containers.
2. Generate a list of morphological elements.
3. For each element, identify the *concepts (entity types and attributes)* that it invokes.
4. Through dynamic interaction with the application, identify the *relationships* between the concepts.
5. Model the concepts and relationships into a semantic network representing the application's *ontology*.

3.1 The Morphological Map

All applications have a morphology, the external interface elements of the system that give its users access to the implemented functionality. In interactive systems, the morphology is the user interface. The components comprising the morphology represent windows or portals through the external "shell" of the application to the underlying ontology. Through systematic interaction with the application's outer shell, we can identify or "excavate" the concepts and the basic relationships between those concepts and model them in a semantic network. In most computing applications, the morphology is the user interface.

We model the user interface in a morphological map. This map consists of the interface's interactive elements or *interactors* (e.g. buttons, text fields, check boxes), *containers*, a morphological element that contains and structures interactors (e.g. windows, dialog boxes, toolbars), and *displays*, morphological elements that present both static and dynamic data about the computing application's states to the user. Figure 1 shows the Windows 95/98 CD Player application along with examples of containers, interactors and displays.

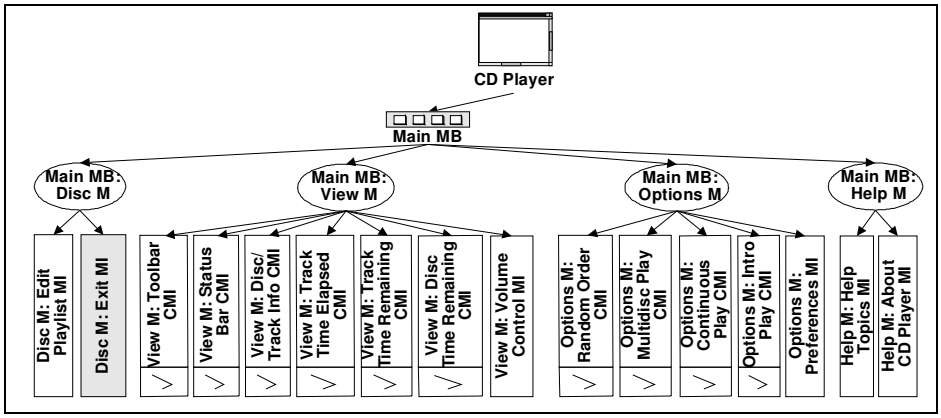
Figure 1 – Examples of containers, interactors, and displays from the Windows 95/98 CD Player



We build this map by traversing and activating all the user interface elements in a systematic, depth-first fashion. Each element is represented by a visual icon and given information

corresponding to its label in the user interface. These visual icons are linked using arrows to show either their container (e.g. a toolbar containing buttons) or their point of activation (e.g. a menu item opening a dialogue box). A portion of the Windows 95/98 CD Player morphology is shown in Figure 2.

Figure 2 – Part of the menu bar in the Windows 95/98 CD Player Morphology.



Morphological elements that can be identified as specific to the computer or operating system running the application are not modeled. These include things such as keys on a keyboard, mouse movements, file handling, or printing capabilities. This also includes all functions and supporting applications that operate independently of the one being studied. For example, the Windows 95/98 CD Player does have a volume command but it activates the Volume Control dialog box of the operating system so we will not model this in the CD Player's morphology. Naturally, if we were excavating the ontology of the operating system itself, we would have to model these things. Ideally, we would like to model only those things within the scope of what we consider to be the morphology of the application being studied; a distinction which can blur when applications begin accessing other items such as web pages.

These elements, their labels, and their interconnections are modeled in a diagram using Microsoft Visio as the drawing tool. Currently, the reconstruction of the morphology into Visio is a manual process.

3.2 Excavating the Ontology

There are text-based ontological notations designed to support data modeling and database exchange activities, such as Ontolingua [80], CLASSIC [35], and CYC [135]. There are also representations for modeling concepts in computing applications, such as entity-relationship (ER) diagrams [17, 25, 46, 76], object-role models (ORM)[88, 159, 200], and object-oriented (OO) diagrams [28, 29, 64, 183]. We have chosen to model our recovered ontologies as a simple semantic network of concepts and relationships [185] [188] [18] [160]. The basic structure of a network, semantic or otherwise, consists of nodes and edges allowing us to use graph algorithms to identify key elements in the ontology. Also, our simple model can be refined into any of the aforementioned models for software developers.

Using the morphological map as an information source, we first identify the concepts indicated by the labels attached to those elements, looking for noun phrases and the indirect objects implied by verbs, a process borrowed from object-oriented analysis methods [28, 183]. For

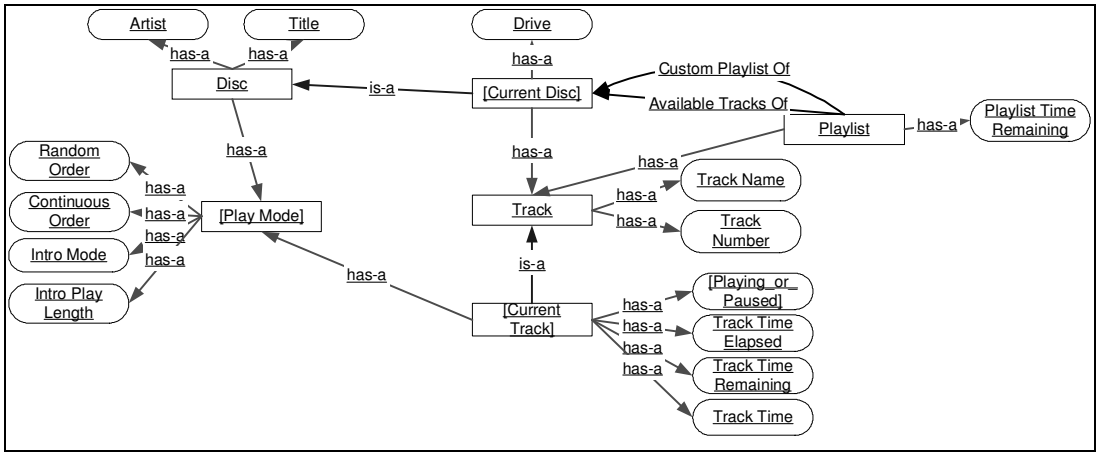
example, a “File Menu” implies that there is a concept of “File”. A “Font” dialog box informs the concept “Font Size”. In cases where a noun does not exist in the label, concept identification requires interaction with the system. For example, “Play” on a CD Player plays a “Track” on a “Disc”. Once we identify a concept, we determine whether it is an entity type, attribute, or instance.

- An *entity* is a thing that can be distinctly identified [46]. A set of entities that share a set of attributes is an *entity type* [61]. Example: In Figure 3, Disc and Track are entity types.
- An *attribute* is an intrinsic property of a thing in the real world [203]. Basically, it is a concept that lacks independent existence except as a property of an entity type. Example: In Figure 3, Track Name and Track Number are attributes of Track.
- An *instance* is a concrete manifestation of an entity type [29].

We model attributes as nodes in our network rather than collapse them into the entity types as one would do in an object model. This is similar to the methods used in NIAM (Natural language Information Analysis Method) [200] and ORM (Object Role Modeling) [87, 159]. For example, a “Disc” in the CD Player has an “Artist” and a “Title” as seen in Figure 3. In our observations of Microsoft Word’s evolution, we noticed several times that concepts that might have been modeled as attributes in one version would become full fledged entity types in the next. Modeling attributes in this manner allows us to make better comparisons of growth and complexity across application versions and examples.

Instances are refined into entity types and do not appear in the ontology. For subtypes, we first identify or name the parent entity type from the common features amongst the subtypes then model it using the ‘is-a’ relationship. These concepts are named using the labels from the morphology. In the cases where the name is unavailable from the morphology or correct modeling requires us to name a concept, we use brackets ([ ]) around the concept name. In the CD Player, we obtained the concept of Disc from the morphology but created the subtype Current Disc to account for the difference between the general concept of CDs and the CD that is currently being played by the application (as shown in Figure 3).

Figure 3 – Ontology for the Windows 95/98 CD Player



After identifying the concepts from the morphology, we identify the relationships between them by interacting with the system and by reconstructing them from observations of both static

information and dynamic behavior. For constructing a semantic network, we use the basic relationships from object modeling: associations, generalization (*is-a*), and aggregation (*has-a*) [29].

- *association* – A structural relationship that specifies that things of one type are connected to things (concepts) of another type. We use associations to indicate an interaction between concepts. In Figure 3, a Playlist can be the Custom Playlist of the Current Disc or displays the Available Tracks of the Current Disc.
- *generalization* – A relationship between a kind of thing (parent or superclass) and a more specific kind of that thing (type, child, or subclass). A *subtype* is a specialization of an entity type. In OO, generalization suggests inheritance where a child inherits the properties of the parent. We use this relationship (and generate the corresponding entity types and attributes) when it becomes clear that several objects have the same characteristics or when we need to infer an entity type to account for an observed behavior. For example, in Figure 3, Current Track is a kind of Track. We chose to model Current Track as the track being played because all Tracks on a CD cannot be played at the same time. Thus, we had to make a distinction between an entity type, Track, that has a name and a number on the CD, and the Current Track, which has a duration and can be played or paused.
- *aggregation* (*has-a*) – A whole/part relationship where one class of entity types represents a larger thing which consists of smaller things. Aggregations are denoted by a 'has-a' relationship in the semantic network. For example, in the CD Player (Figure 3), a Playlist possesses the concepts Track and Playlist Time Remaining. In our modeling conventions, we break the traditional convention of requiring both things to have independent identities in the case of attributes. However, attributes are not permitted to have has-a relationships with other concepts.

In other diagrammatic methods, we would also show constraints on these relationships such as cardinality (the number of elements in a set) or dependencies (a semantic relationship between two things) [17, 28] but we chose not to model them because they were not essential to our analysis. We also model these relationships in our diagrams as directed edges between the concepts for the sake of readability but do not use digraph algorithms in our analysis. It turns out that modeling ontologies as a directed graph produces too many isolates and components that confound the analysis. It was also not clear from our survey of modeling methods that the directed arrows were meaningful from a graph or network standpoint and served only to enhance readability for software developers. Currently the process of excavating the ontology from the morphology and producing the ontological diagram in Microsoft Visio is a manual process.

4 Ontological Analysis

To analyze ontologies, we identified techniques from graph theory, specifically those used in social network analysis [205] to identify the following items:

- *Core concepts* and *peripheral concepts*; a core concept is essential to the application's ontology while a peripheral one is not. We identify these using *node betweenness centrality* measures [205].
- *Teleons*; Since a teleon is a collection of related concepts, we believed that their structural dependencies will be evident in the graph that represents the ontology. We identify teleons using a *k-core analysis*.

We wrote a Visual Basic macro for Visio that refines a graph into an adjacency matrix that could be read by an application called UCINET, a tool for social network analysis [31]. The social and behavioral science communities model relationships between social entity types as social networks. Social network analysis methods apply algorithms from graph theory to identify both patterns and variables in the structural relationships of these networks [205]. By treating the concepts in our semantic networks as social entities in a social network, we can use the tools in UCINET for our analysis.

4.1.1 Core Concepts and Betweenness Centrality

In our Microsoft Word study, we observed that older concepts tended to remain unchanged through successive versions. This implied that certain concepts may be integral to that application. Remove the 'paragraph' or 'word' concept from a word processor's ontology and you may have crippled its ability to implement many features. Remove something like 'hyperlinks' or 'graphics and you may not be able to compose web pages but you can still generate other types of documents. We call the integral concepts *core concepts*. In our semantic network, we believed that the core concepts would be ones that are the most central in the graph.

Prestige or *centrality measures* in social network theory are used to identify individuals who have importance relative to the other members of the network. Betweenness centrality measures the number of shortest paths that use a particular node. It computes a normalized value between 0 and 100 where 100 means that the node lies on all shortest paths between all pairs of nodes. In our semantic model, a path between two concepts indicated a potential interaction or dependency at the ontological level. Naturally, the longer the path, the lower the probability that the two concepts are related. But over time, certain concepts can migrate in the ontology or develop new relationships to nearby concepts. An example of an evolved relationship can be found in the Win 95/98 CD Player where a Disc has a Title and an Artist while a Track has a Track Name. Tracks, therefore, do not have Artists. However, in modern media players, tracks can actually be assigned different Artists names while a Current CD being transferred to MP3 files still has an Artist.

Other centrality measures from social network theory include:

- *Degree Centrality* measures the number of edges on a node. The more edges on a node, the higher the centrality.
- *Closeness Centrality* measures the average distance from that node to all other nodes. The closer the node to all other nodes, the higher its centrality.

- *Information Centrality* measures the information contained in all paths originating with a specific node.
- *Eigenvector Centrality* measures the centrality of a node relative to the importance of its surrounding nodes.

We applied these measures to at least three different ontologies and determined that betweenness centrality measures not only had the lowest sensitivity to small errors in the model but returned what we considered to be the most intuitively correct core concepts in the graph. It also had the additional benefit of ignoring attributes (modeled as leaf nodes) in the analysis.

Figure 4 – A comparison of centrality measures on the concepts in the Notepad ontology. Concepts were sorted in ascending order based on the values returned by the method – y-axis is the normalized centrality value. The x-axis would normally have been the numeric label for the concepts but, since the data sets were sorted independently, it is meaningless in this graph.

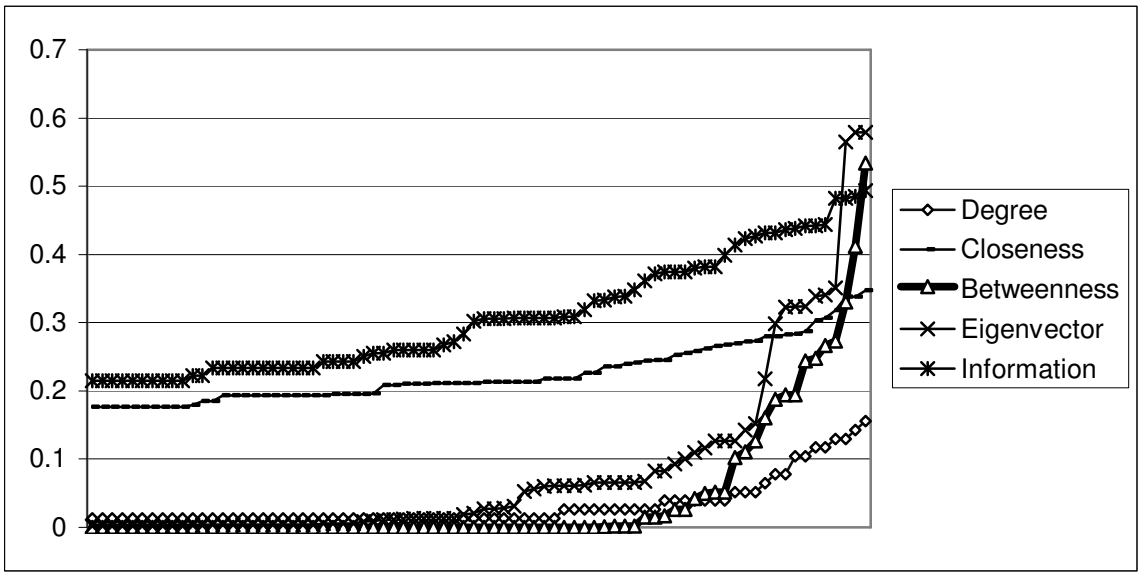


Figure 4 shows the results of one of our comparisons. Knowing from visual inspection of the graph that some concepts should have a higher prestige than others, we were looking for a characteristic curve that would allow us to determine a threshold centrality for deciding where core concepts began. Degree and closeness centralities produced very flat measures across all concepts. While betweenness, eigenvector, and information centrality plots showed distinct curve, we also saw in our analysis that betweenness tended to return concepts that made more sense. For example, using the eigenvector centrality measure on the Palm Pilot scheduler showed that Start Time and End Time were more important concepts than Time or Alarm. Because the eigenvector centrality of a particular node depends on the importance of its neighbors, that node can develop an enhanced measure simply by being incident upon several prominent nodes. Information centrality had similar problems with proximities to prominent nodes. We chose betweenness centrality as the best measure for identifying core concepts in a graph.

4.1.2 Teleons and K-cores

From our Microsoft Word study, we knew that certain concepts were more interrelated than others. For example, Table, Column, Row, and Cell had a strong conceptual interrelationship. What we wanted was a method for identifying tightly related subgraphs of nodes in our ontology. These types of subgraphs have been described in graph theory and include *cliques* (a maximal,

complete subgraph of three or more nodes) and *n-cliques* (a maximal subgraph in which the largest geodesic distance between any pair of nodes is no greater than n) [49, 184]. It turns out that the mathematical restrictiveness on subgroup membership did not lend itself to obtaining meaningful results on our ontologies. For example, Notepad would return five different cliques which were permutations of Alignment, [Header/Footer Code], and Left / Right / and Center Alignment Codes. We decided to use a *k-core analysis* [205]. A *k*-core is a maximal, induced subgraph such that each node in the subgraph has edges connecting it to *k* or more nodes. Wasserman and Faust describe it as useful for exposing regions of the graph where you were likely to find other subgraphs such as cliques. We hoped the k-cores would reveal teleons similar to those we identified in our previous study.

4.2 Case Studies of Ontological Excavation

We applied our models and techniques to four applications: Windows 95/98 CD Player, Palm Pilot Scheduler, Microsoft Notepad, and the Protocol Calculator/Calendar device (see Appendices 2, 3, 4, and 5 for descriptions and data from our studies).

4.2.1 Core Concept Identification

After examining our data, we settled on an arbitrary betweenness threshold of 7.0 as the cutoff point between core and peripheral concepts. In several of our case studies, that value seemed to be the difference between a core concept and a concept that supported a core concept. We do not claim any significance to that value as yet and acknowledge room for debate. More empirical studies may confirm or disprove the usefulness of this threshold. Nevertheless, we were able to identify the following core concepts from these applications:

Table 2 – Core Concepts found in the case studies. Concepts are listed in order of their betweenness centrality values

Application	Candidate Core Concepts
CD Player	[Current Track], [Play Mode], Track, Disc, [Current Disc], Playlist
Palm Pilot Scheduler	Event, Date, To Do Item, Hot Synch, Day, Month, Time, Alarm, Repetition, Note, Every
Notepad	Page Setup, Font [Setting], Paper, Text, Paper Size, Font, Script, Header, Footer, [Configuration], [Header/Footer Code], Margins, Alignment, Font Style
Protocol Calculator / Calendar	[Time Zone], Time, Home Time

The Protocol Calculator / Calendar actually had multiple components (isolated subgraphs) in its ontology. We performed a separate analysis on each subgraph and obtained the following core concepts per subgraph:

Table 3 – Core concepts found in Protocol Calculator / Calendar. Note: Subgraph 4 only has 2 nodes.

Subgraph	Core Concepts
1	Date, Month, Year, Calendar
2	[Time Zone], Time, Home Time, [Time Display Mode], Alarm Time, Alarm
3	[Mathematical Operation]
4	Currency Exchange [Calculator], Exchange Rate *

We believe that our use of betweenness centrality to the excavated ontologies succeeded in identifying candidate concepts that could be argued to be core concepts in their respective applications. In the case of the Protocol Calculator / Calendar device, we identified four independent subgroups in the ontology and only three core concepts. Within those subgroups, our analysis revealed core concepts that define each of them respectively.

4.2.2 Teleon Identification

The k-core analysis was able to identify the following subgroups in our sample applications:

Table 4 – CD Player Teleons Identified by K-Core Analysis

k-value	Concepts in Subgroup
2	[Current Track], [Play Mode], Track, Disc, [Current Disc], Playlist

Table 5 – Palm Pilot Scheduler Teleons Identified by K-Core Analysis

k-value	Concepts in Teleon
2	PurgeUnits, Week, Month, Every, Today, Day, Preferences, End Date, Repetition, Schedule, Year, Date, Due Date, All Occurrences, Current Occurrences, Event, To Do Item, Is Private, Start Time, End Time, Alarm, Alarm Units, Hour, Minute, Backup Copy, Note, Event Problem, Synch Problem, To Do Problem, To Do List, Hot Synch, Time, Application

Table 6 – MS Notepad Teleons Identified by K-Core Analysis

k value	Concepts in the core
3	Text, Header, Footer, File Name, Page, Number, Date, Time
2 (a)	Header/Footer Code, Left/Right/Center Alignment, Alignment (of Header/Footer)
2 (b)	File, Current File, [Configuration], Line, Word, Font [Setting], Page Setup, Document, Page

Table 7 – Protocol Calendar / Calculator Teleons Identified by K-Core Analysis

k-value	Concepts in Teleon
2	Date, Month, Year, Calendar
2	Alarm, Alarm Time, Count Down Timer, Hour, Minute, Second, Sound, Time,

The CD player's 2-core consists of those concepts which are entity types in the ontology. The Scheduler's ontology has such a large number of concepts in its 2-core to the point where it is nearly indistinguishable from the ontology itself. The Notepad and the Calendar / Calculator ontologies produced interesting results. The Notepad ontology had one large cluster which could be broken into 3 distinct k-cores. The concepts in the 3-Core all concern Text, which we expected would be the case since MS Notepad is a text editor. The concepts in the 2-cores involve Header/Footer Codes and File Handling / Document Format, respectively. Thus, the k-core technique shows that MS Notepad has Text, Header / Footer Management, and File Handling and Document Formatting features. The Calendar / Calculator device had two 2-cores related to the calendar date and timekeeping. However, it did not show that the device also had a countdown timer, a calculator and a currency exchange calculator.

While k-cores did not reveal teleons at the granularity that would have revealed features at the morphological level, it did show interesting relationships amongst the concepts in these applications.

5 Conceptual Coherence and Ontological Structure

From our studies of Microsoft Word and from our case studies of these small applications, we have developed techniques for excavating the ontology of a computing application and for analyzing these ontologies. We have proposed that conceptual coherence measures the degree to which a computing application's concepts are tightly related and that it can be used to assess the application's usefulness. In this section we will describe and motivate these ideas further. We will also show potential metrics for measuring conceptual coherence and the results that we obtained from our case studies.

5.1 Coherent Applications

The classical view of concepts, derived from Aristotelian philosophy, asserts that concepts can be defined by providing a set of necessary and sufficient attributes that belong to that concept. However, this classical view failed to take into account functional features or disjunctive concepts. [185]. In addition, Wittgenstein argued that concepts cannot be defined concretely by their features because many of these concepts have not been concretely defined and therefore lack these defining features. Using empirical examples such as "games", he showed how games could be difficult to categorize by their attributes. For example, the concept of a "game" for a professional athlete has very different attributes than the concept of a "game" to a child. An athlete sees games as a job and "playing the game" as work. A child sees games as entertainment and "playing the game" as fun. Therefore, the classical view proves inadequate in categorizing concepts [208].

Nevertheless, concepts in computing applications are engineered using this classical view of concepts. These concepts are designed and implemented with a specific set of attributes. Within the bounds of an application's ontology, concepts have a very specific design and meaning. However, their implementation can have many variations and mutations. A concept with the same name in a different application may have very different attributes. In addition, their equivalent concepts in the real world often lack concrete definitions and can change meanings depending on the use context. Categorizing applications into specific families by their common features can be difficult because of these variations. Certainly, broad categories are possible, such as Jackson's problem frames [105]. Jackson describes broad categories of applications by their objectives. For example, the workpieces problem frame characterizes situations where a workpiece (a piece of material) is being modified by a tool to produce a desired artifact. Specific categories within these broad categories are harder to qualify.

Some application families have such a small set of features that they are easy to define concretely. The Windows 95/98 CD Player does not have functions having to do with anything other than managing the playing of CDs. The Windows 2000 CD Player adds features for downloading track titles from the Internet. If we added the ability to play MP3 files or download them from the Internet, then intuitively, we may argue that it is no longer just a CD Player. It has become less coherent. Other application families have such a large set of features that they are immediately difficult to define concretely. For example, if we wished to characterize a word processing family we might identify a candidate word processor and determine its essential concepts as a first approximation. In our evolution study, we saw that all versions of Microsoft Word possessed features that concern text editing and formatting, things that would belong to a domain that we might call word processor. However, later versions included graphics, graphical

layout, HTML editing, and Web page controls. The current version of Microsoft Word can be argued to still have the essential features of a word processor but also the features of a Web page editor and a desktop publisher. Its conceptual coherence has decreased over time because of its multiple feature sets. This decreased conceptual coherence has contributed to its users' perceptions of bloat.

5.2 Metrics for Coherence

While we believe, intuitively, that conceptual coherence is a property of all applications, we would like a means for quantifying it in our ontologies. We have identified the following candidate metrics that can be used to measure the conceptual coherence of applications.

- *Core Concept Percentage* – We expect that an ontology with a disproportionate number of core concepts relative to the rest of the ontology has a structure possessing many interdependencies amongst the concepts.
- *Average Centrality* – Our centrality measures identify those concepts with a high number of dependencies. However, an ontology can have many concepts on the periphery, with some ties to core concepts, that have a non-zero centrality value but not high enough to be considered as a core concept. These may be supporting concepts. An ontology with a small number of core concepts but a large number of concepts that directly support core concepts may have a high coherence.
- *Network Density* – Density is the number of edges in a graph divided by the number of possible edges. Since edges in our graph signify relationships between concepts, then a graph with a large density may indicate a higher conceptual coherence.

We applied these metrics to the applications chosen for our case studies and found the following (Table 8):

Table 8 – Comparison of Coherence Metrics

	Core Concept %	Core Concept % (w/o attributes)	Average Centrality	Network Density
Windows 95/98 CD Player	30	100	9.60	.11
Palm Pilot Scheduler	17	23	4.22	.05
Microsoft Notepad	18	19	4.83	.03
Protocol Calculator / Calendar	6	6	1.97	.04

The CD Player shows the highest values across all metrics which seems to indicate that it has the highest conceptual coherence while the Calculator / Calendar has the lowest values. Intuitively, this would seem to make sense as the CD Player just plays CDs while the Protocol device tells the time in sixteen different time zones and has a countdown timer, alarm, calculator, and currency exchange calculator. We also looked at several different versions of the Notepad ontology summarized in Table 9.

Table 9 – Comparison of Coherence Metrics across versions of the Notepad Ontology

	Core Concept %	Core Concept % (w/o attributes)	Average Centrality	Network Density
Microsoft Notepad, v.1	18	19	4.48	.03
Microsoft Notepad, v.1 (No Paper or Scripts)	20	21	4.83	.04
Microsoft Notepad, v.2	21	25	5.42	.03

We first collected metrics on Notepad's ontology then removed concepts related to types of Paper and types of Scripts because we were testing how a concept with many subtypes on the periphery could affect the overall ontology. The second version shows a higher conceptual

coherence across all three measures. We then reconstructed the ontology, correcting some of our modeling assumptions. The second version of Notepad's ontology has 26 fewer concepts and attributes than the first version and shows a higher conceptual coherence than either of the other versions. (Details on how the second version was derived can be found in Appendix 6.)

In our first modeling pass, we decided to be strict about how we applied black box methods and in the absence of information from either the user interface or from the help files, we simply modeled items that we found in a list of scripts and paper sizes as concepts in the ontology. Since no other concept depended on them, they ended up being modeled as leaf nodes, giving their parent concepts, Script and Size, a high centrality value. When we removed these peripheral concepts, Notepad's overall conceptual coherence increased as we predicted. In the second version of the ontology, we made a number of corrections to our modeling assumptions. In spite of the large number of concepts removed relative to the ontology, we still identified most of the same core concepts and the same teleons. What our reification did accomplish, according to our metrics, is produce an ontology with a higher conceptual coherence than the original.

We next studied the Protocol Calculator / Calendar device in depth. The results of this comparison can be found in Table 10.

Table 10 – Comparison of Coherence Metrics across ontologies of the Protocol Calculator / Calendar (* The Currency Exchange Calculator subgraph only has 2 nodes)

	Core Concept %	Core Concept % (w/o attributes)	Average Centrality	Network Density
Protocol Calculator / Calendar	6 %	6 %	1.97	.04
Protocol Calculator / Calendar (No Time Zones)	6 %	6 %	1.34	.06
Calendar Subgroup	80 %	80 %	20.00	.50
Time Subgroup	20 %	20 %	7.99	.07
Time Subgroup (No Time Zones)	43 %	43 %	13.64	.18
Calculator Subgroup	10 %	10 %	9.09	.18
Currency Exchange Calculator *	100 %*	100 %*	0.00*	1.00*

After analyzing the excavated ontology, we generated a modified version of the ontology where we removed the time zones as concepts and found that the average centrality actually decreased across the application while the density increased. This implied that the entire ontology became less coherent with the removal of these concepts. However, because the ontology actually had four different subgroups, We then applied the metrics to each of the four distinct subgroups of the device's ontology, performing a second analysis on the Time subgroup without the time zone concepts. Each of the subgroups showed a much higher coherence than the overall application with the Calendar Subgroup showing the highest coherence of any of the ontologies examined so far. The Time Subgroup also showed a higher coherence without the time zone concepts. The Time subgroup not only contains the basic concepts for expressing time but the concepts for setting the alarm and the countdown timer. Thus, we would expect it to have a lower conceptual coherence than the Calendar subgroup but still have a high conceptual coherence since all its concepts are still fundamentally tied to timekeeping. The Currency Exchange Calculator ontology is a strange case because it only has two concepts in its subgraph but is included for the sake of completeness.

We have claimed that high conceptual coherence correlates to usefulness. However, we still lack the data to be able to say exactly what thresholds for any of the metrics indicate a high

coherence. While the CD Player and individual subgraphs within the Protocol Calculator / Calendar device show higher values for conceptual coherence relative to the other applications, and we are comfortable claiming from intuition that these applications are conceptually coherent, we still cannot say whether the other applications have a high or low coherence. We suspect that they are coherent but possess a number of peripheral features that keep them from exhibiting high coherence. Specifically, the Palm Pilot Scheduler has many concepts for synching the data with an external application and for backing up the data, and Notepad, ostensibly a text editor, has many concepts related to printing. With more empirical studies, we may be able to qualify these metrics further and identify which one of the candidate metrics measures conceptual coherence the best. Correlating high conceptual coherence to usefulness will require a separate set of empirical studies.

5.3 Ontological Structures

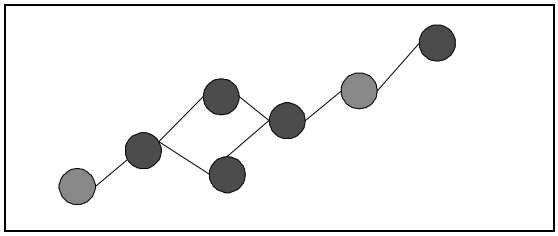
From our case studies of ontological excavation, we have observed some distinct characteristics in the recovered ontologies, such as the CD Player's central cluster of core concepts surrounded by attributes, the Protocol Calculator / Calendar's multiple subgroups within its ontology, and Notepad's multiple teleons. We also noticed a possible correlation between these characteristics and the conceptual coherence metrics for their respective applications. These observations suggest that applications may be categorized by their ontological structures. These structures exhibit common forms that likely emerge from design and evolution. We present three hypothetical archetypical ontological structures: the Reef Structure, the Toolbox Structure, and the Urban Structure.

5.3.1 The Reef Structure

A Reef structure represents a system that implements a tightly related set of concepts, possessing a high conceptual coherence. Many metaphors exist that could express the idea of a unified set of concepts constructed around a central architecture. We chose a biological one to account for the evolutionary behaviors that we have observed in the ontologies of single purpose applications. Coral reefs are ecosystems built on a skeleton of calcium deposits created by tiny creatures called coral. This skeleton provides a habitat for many species that each play a role in maintaining the reef environment. Over time, the reef can grow and develop a very rich and stable biological system [14]. A Reef ontological structure has a central structure that not only supports itself but also a number of other entity types that contribute to the overall system.

A Reef-like computing application has an endoskeleton of core services and layers of peripheral services that are supported by the endoskeleton. For example, a spell-checker for a word processor is a peripheral service. Spell checkers could not exist independently of changes for text that has to be spelled correctly. Figure 5 shows an abstract ontology for what we would expect to find in a Reef Structure application – a skeleton of core concepts with some supporting ones on the periphery.

Figure 5 – Reef Structure of Conceptual Coherence

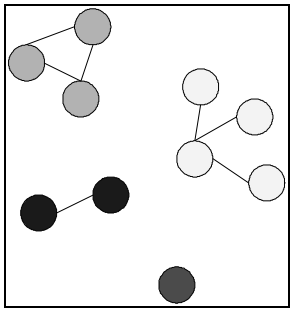


Simply stated, this application does one basic thing. It may have services that aid in the achievement of that central goal but these services could be removed without much loss to the overall application. Arguably, most applications begin as reefs – tools built for a single purpose. Over time, applications with well-defined goals or constrained domains may evolve by acquiring new features but they only tend to acquire those features that can directly support and enhance the existing ones. We believe that examples of Reef workpiece applications include PowerPoint (essentially just a presentation generator), Adobe Photoshop (a bitmap editor), and TurboTax (a financial management tool designed to produce tax documents). Most small applications and games, like CD Players, Solitaire, Calendars, Clocks, and so on are also Reef structures. An example of a Reef ontological structure can be found in Appendix 2 which describes the case study for the Microsoft Windows 95/98 CD Player application.

5.3.2 The Toolbox Structure

A toolbox that you might find in a home is a collection of tools that have different affordances for specialized tasks. Hammers, screwdrivers, and pliers all contribute to different sorts of tasks but one usually does not use every tool in a toolbox to accomplish a task. A Toolbox structure (Figure 6) has a collection of conceptually unrelated and lightly related ontologies that have been assembled for reasons of convenience or design under a single morphology.

Figure 6 – Toolbox Structure of Conceptual Coherence

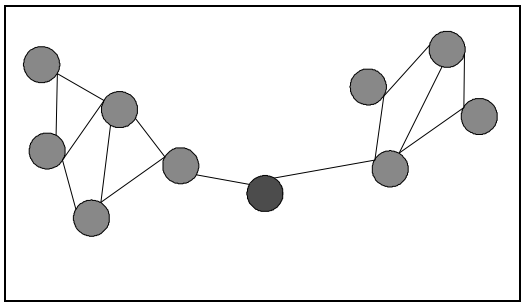


The tools in a toolbox collectively support a broader category of goals such as resource management, information management, or media playing. Over time, a Toolbox may collect more tools, enhance the capabilities of its existing tools, or begin merging the tools by combining their functions. Examples of Toolbox model workpiece computing applications include RealOne Player (a media player that supports CD playing, CD burning, Internet radio, web browsing, and MP3 management) and Yahoo! Instant Messenger (ostensibly a instant messaging tool that also delivers information such as weather, stocks, auctions, and news). The overall Toolbox structure will have less conceptual coherence by definition but will be structurally coherent within each individual tool. An example of a Toolbox ontological structure can be found in Appendix 4 which is a case study of the Protocol Calculator / Calendar device.

5.3.3 The Urban Structure

Urban areas in cities are often divided into large neighborhoods that compete for influence, income, resources, business, and desirable populations of people. Sometimes neighborhoods will fragment into smaller zones. Other times, neighborhoods will subsume other less successful neighborhoods. The collective urban environment may be easy to identify on a map but the subordinate areas within that region may not be.

Figure 7 – The Urban Structure of Conceptual Coherence



The Urban Structure (Figure 7) results when an application has acquired features that cause its ontology to lose conceptual coherence. Large clusters of features may merge, blurring the boundaries between some services, or fracture, causing others to become more isolated and independent of the computing application. We expect the ontology of an Urban application to contain competing clusters of core concepts (multiple and unrelated teleons that have similar sizes and influences on the ontology). It differs from the Toolbox Structure in that these core concepts are connected to each other. A Toolbox can have large independent clusters of ontologies because, in practice, each cluster represents a tool that is used independently.

The Urban Structure can result from a design that had poorly articulated or confused requirements. It also might have begun as a Reef or Toolbox model but over time had evolved by growing in size and functionality to acquire new customers that have different and occasionally conflicting, requirements for this application. Over its lifetime, such an application may be perceived to be more bloated by users who find themselves using smaller and smaller percentages of the overall system with each release. Examples of Urban workpiece computing applications include Microsoft Word and Microsoft Excel. An example of an Urban ontological structure can be found in Appendix 5 which describes the Microsoft Windows Notepad application.

5.3.4 Ontological Structures and Computing Applications

While still hypothetical, these archetypical ontological structures have interesting implications for aiding designers in designing and enhancing their applications. If this and future research show a correlation between usefulness and conceptual coherence, then one could imagine design heuristics encouraging adoption of the Reef or Toolbox ontological structures as a framework for organizing domain concepts in the ontology of an application prior to developing the software architecture. Bloat could be detected in the ontology by detecting Urban ontological structures in existing applications and in the ontologies of future application versions before proposed features are added. In applications with a close correspondence between its concepts and the underlying software architecture, software maintenance activities could include *ontological grafting* and *pruning*; adding teleons to the ontology or removing them to preserve conceptual coherence. For example, if a computing application is found to have an Urban ontological structure, one could

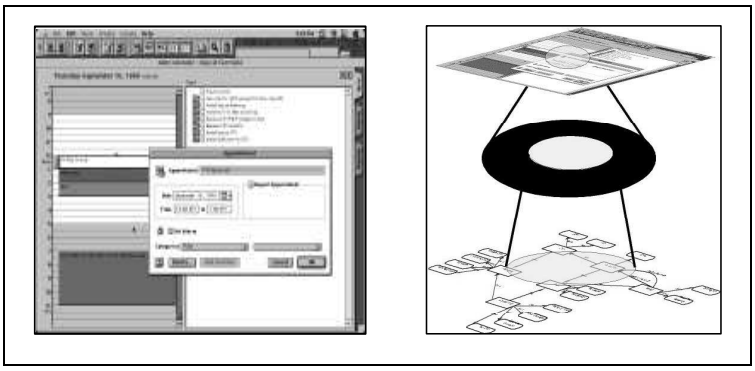
preserve the stability and enhance the maintainability of the application by pruning one of the competing clusters of core concepts and creating a separate application that contributes services without sharing morphologies. These potential represent tremendous payoffs in improving software usefulness and in reducing unnecessary effort at the development stage.

6 Use Case Silhouettes

We know that a gap exists between potential and actual usage of an application. An application can be conceptually coherent in its ontology without being useful in a given context. However, we believe that applications that have a high conceptual coherence will have a higher conceptual fitness than a similar application with a low conceptual coherence *assuming that their features are appropriate for that use context*. This is simply because the application with high conceptual coherence is less likely to have concepts that go unused in that particular use context. To examine the relationship between conceptual coherence and usefulness relative to a specific use context, we have developed a technique called *use case silhouetting* that takes a set of use cases and measures the amount of *ontological coverage* by those use cases.

When the services of a computing application are engaged by its users, it uses or invokes concepts in its ontology. Ontological coverage measures the percentage of the ontology covered by those concepts for the desired unit of analysis. For example, one could measure the amount of ontological coverage for a given user's actions, a scenario, a goal, a specific task, an organization, and so on. We can also measure the importance of these individual concepts by examining the frequency by which they are activated. If we find that the unit of analysis has a high ontological coverage, we could infer that the application has a high usefulness insofar as its conceptual correspondence is concerned. We refer to the process of collecting data on concept frequency as *silhouetting*.

Figure 8 – The Silhouette Metaphor



The application's morphology (e.g. the user interface) provides affordances that permit access to the services. Viewed another way, these morphological elements provide portals in the 'skin' of the application through which the underlying conceptual model can be seen. Activating particular elements in the morphology casts a 'silhouette' on the concepts below where only specific concepts are highlighted as we show in Figure 8. While any of the units of analysis that we have mentioned would be appropriate for measuring ontological coverage, we have selected *use cases* as our source of data.

Use cases come from the Unified Software Process where they are used to express requirements and guide developers in the design, construction, and testing of the system [109].

> *"A use case specifies a sequence of actions, including variants, that the system can perform and that yields an observable result of value to a particular actor." [109]*

In software development, specifically in the requirements phase, the developer will gather narratives from users and structure them into these use cases. By using *use case silhouettes*, we can measure the ontological coverage for a proposed system for each use case and for a set of use cases that reflect a specific scenario, user, or set of requirements.

A use case silhouette is developed by recording the number of times a concept is referenced in a particular use case. This can be done in a number of ways. For high level use cases, where the interface is not mentioned, we can simply examine the concepts activated at each step of the use case. For example, a use case action that says "The customer requests a transaction slip from the system." tells us that the 'customer', 'transaction', and 'transaction slip' concepts have been activated by the as-yet nonexistent user interface. For low level use cases that explicitly describe how the user interface is activated, we simply account for each morphological element mentioned in the sequence of actions and trace the concepts that they invoke. For example, a use case action that says "To change the size of a character, on the Formatting toolbar, click a point size in the Font Size box." invokes the concepts 'character', 'font', and 'point size' through the morphological elements Formatting Toolbar and Font Size Drop Down Box.

Here is sample text from one of the use cases of the Windows 95/98 CD Player (CDs: storing track titles):

> To store the track titles of your CD
>
> 1. Make sure your CD is in the drive.
> 2. On the **Disc** menu, click **Edit Play List**.
> 3. In **Artist,** type the artist's name.
> 4. In **Title**, type the title of your CD.
> 5. In **Available Tracks**, click the track whose name you want to store.

From this use case, we identified the morphological elements Disc Menu, Edit Play List Menu Item, Disc Settings Dialog Box, Artist Text Field, Title Text Field, and Available Tracks List. From these elements we identified the concepts Disc, Artist, Title, Track (2 times), Track Number, and Playlist (3 times).

We can learn the following from use case silhouettes:

- *The total amount of ontological coverage provided by a set of use cases.* – Assuming that the use case set provides a complete set of usages by a user or a specific use context, what is the percent of ontological coverage reached? If the coverage is low, then the application may not have high usefulness for this set of use cases.
- *The parts of an ontology that are covered by those use cases and to what degree.* – A set of use cases may emphasize certain parts of an ontology over others. Even though all concepts may eventually be engaged, some concepts may see more silhouetting than others. These may correspond to the core concepts of the application or indicate concepts that are important only to that set of use cases.
- *The amount of ontological coverage by a particular use case.* – An individual use case may have low or high engagement with the application's ontology, measured by the number of concepts, especially core concepts, that it activates. A frequently used use case with high engagement with an application must be considered carefully during design

because the concepts that it uses could affect the overall ontology even if those concepts lack importance by the structural measures of centrality.

- *The importance of a particular concept relative to a set of use cases.* – A concept frequently invoked by the use cases may or may not have structural importance in the ontology. In either case, its design will impact the performance of those use cases.

Each use case describes a goal that the user wants to achieve and the sequence of actions performed on the morphological elements of an application required to achieve this goal. Because ontological excavation links each morphological element to a set of concepts, we can count the concepts activated across all the use cases to collect statistical data of activation frequency. This data allows us to measure both general and specific ontological coverage. General ontological coverage looks at how many concepts in the ontology were activated by a set of use cases. Specific ontological coverage examines how often each concept was activated by a use case to determine a concept's relative importance in the ontology for that given set of use cases.

Examples of use case silhouettes and their analysis can be found for the Windows 95/98 CD Player, Microsoft Notepad and the Protocol Calendar/Calculator in Appendix 2, 4, and 5. No use cases were available for the Palm Pilot Scheduler.

7 Research Framework

We summarize our research framework for studying the usefulness of computing applications, the abstraction that we will use to model them, and the hypotheses which we wish to investigate.

We begin with the following claims:

- Features are the *user-accessible behaviors and services implemented by a computing application*. We distinguish these from system-level modifications and services which implement or optimize the behaviors at the user level but are not seen or accessed by the user.
- Computing applications have a view of the domain which is expressed by its features to its users. This theory consists of concepts and relationships that we model in an *ontology*.
- An application's ontology is accessed by its users through its user interface. In this work, we call this interface the *morphology* of an application.
- Each application has a set of concepts essential to defining that application's feature set and identity. We call them *core concepts*. All other concepts are considered to be *peripheral* in the ontology.
- Ontologies have subgroups of strongly related concepts which we call *teleons*.
- Usefulness is a function of an application's conceptual fitness – the degree to which an application's ontology matches the domain of the user.

We use the following abstractions to model computing applications:

- An application model consists of a morphology and an ontology where the elements that compose the morphology serve as portals to the underlying concepts and relationships of the ontology.
- We model an application's morphology as a graph of user interface interactors, displays, and containers where edges trace paths of access and activation.
- We model an application's ontology as a semantic network similar to an entity-relationship diagram. This graph models the concepts, consisting of *entity types* and *attributes*, as nodes and relationships modeled as edges.

We analyze the ontologies to identify the following:

- *Core concepts* – concepts that play a prominent role in the structure of the ontology.
- *Teleons* – tightly linked subgroups of concepts.

We also develop use case silhouettes on our applications to identify the following:

- The ontological coverage of a use case set – the percentage of concepts activated in the ontology by the use cases.
- The concepts of the ontology that are activated by the use cases.
- The importance of each use case relative to the concepts it invokes.
- The concepts that are important to those use cases.

We make the following claims.

- Application ontologies have a quality that we call *conceptual coherence*. Conceptual coherence measures the degree to which a computing application's concepts are tightly related.
- Applications, based on how they were designed and evolved, have growth patterns that produce an ontological structure that affects their conceptual coherence. We conjecture three archetypical structures: Reef, Toolbox, and Urban.
- Applications with high conceptual coherence are more likely to be perceived as useful to their users.
- Application usefulness can be measured with a combination of conceptual coherence measures, for a general approximation, and use case silhouettes, for an approximation relative to a specific use context.

Our central thesis is:

- The conceptual coherence of an application determines its perceived usefulness to its users and features with only tangential relationships to those features essential to an application are less likely to be used and reduce that application's conceptual coherence.

We will now propose three studies to study these claims. Because of the theoretical nature of this work, the first two studies will be exploratory where we will gather empirical data on some large systems. Our last study will be confirmatory where we will use usability data gathered and validated by another researcher to validate some of our claims.

8 Proposal

We plan to undertake the three following studies to further explore and validate our models of computing applications, their features, and underlying ontologies.

8.1 Study 1: Recover ontologies for three large and evolved systems.

Our first study will be exploratory and will serve two basic purposes: to test the scalability of our methods and to supply us with data that we can use for later study. We will excavate the ontologies for three different systems. Each of these systems was selected for its feature complexity, evolutionary age (number of versions that it has generated), and observed resemblance to one of three ontological structures that we proposed in Section 5.3: Reef, Toolbox, and Urban. Because we were going to choose the systems by our intuitions about their underlying conceptual structure, we identified characteristics that could be used as normalized selection criteria to avoid further bias in our analysis. The systems we chose have similar ages, evolutionarily speaking, and thus have feature sets with sizes and complexities commensurate with this age. We have also chosen well-known commercial off-the-shelf (COTS) applications that belong to the workpieces problem frame [105] so that they have conceptual continuity in their external presentation and in the features that they provide. We have chosen to study the following systems:

A) *Microsoft Word 2000* – Word 2000 is a logical choice for our current studies because despite having many versions in its history, it still remains one of the more popular examples of bloated software ([74], [144-146]). Our previous study surveying three of those versions showed a central cluster of concepts consisting of text processing features overlaid by newer features which include graphics and Internet support capabilities [98]. We anticipate that Word 2000 has low conceptual coherence and resembles an Urban ontological structure.

B) *Microsoft Powerpoint 2000* – Powerpoint 2000 is primarily a tool for giving presentations making it a potential Reef structure. While it does possess mechanisms that can be exploited to other ends (we have seen Bob Balzer turn Powerpoint into a software development environment by using .COM listeners [16]), we believe Powerpoint will reveal an ontology that contains a primary cluster of core concepts related to presentations and slides and supporting concepts such as graphics and animations as smaller ontological clusters connected to it – characteristic of a Reef ontological structure. While there are many single purpose applications on the market, we have also chosen Powerpoint because, like Word, it is also published by Microsoft. Microsoft Office products have been collectively derided for bloat and feature creep so, in anticipation of criticisms that we only chose Microsoft Word, the company's most famous product for 'bloat' as a straw man, we include another Microsoft product which we anticipate will be found to have high conceptual coherence.

C) *Yahoo! Instant Messenger 5.6* – Instant messaging applications, programs that allow two or more users to send text messages to one another instantaneously, have been around almost since the inception of the Internet. Over time they have become much more sophisticated, both in implementation and delivery, and more diverse in the information that they deliver. Yahoo! IM is in wide use and has developed a number of extra features such as Weather, News, Stock Alerts, and Sports. It also has the benefit of being relatively simple in structure and unencumbered by advertisements unlike ICQ, AOL Messenger, and MSN Messenger. It has groups of features that

are both functionally and conceptually distinct from each other, making Yahoo! IM a potentially good example of a Toolbox application.

In addition to the empirical data that this study will provide to answer later questions, we wish to answer the following exploratory questions:

- Do these systems reveal distinct ontological structures?
- Do these large systems reveal teleons that map to known features?
- What do the ontologies reveal about the conceptual coherence of these applications?

8.1.1 Ontological Structure Identification

We have claimed that applications can be mapped to at least three ontological structures but reached this number from observation and logic. While our earlier small case studies showed at least one ontology, the Protocol Calculator / Calendar, that resemble a Toolbox, we have yet to show convincingly that the CD Player is a Reef or that Notepad is an Urban Structure or that there is any true structural differences between the two. We would like to identify the structural characteristics of ontologies that would allow us to map them to these structures.

8.1.2 Teleons and Features

Some of our intuitions that led to our work on teleons came from observing the conceptual relationships in features like Tables and Text in word processing. We would like to see whether extracting teleons from these large applications will reveal tight clusters of related concepts that match known features at the user level.

8.1.3 Conceptual Coherence

We have identified some preliminary thresholds for high and low conceptual coherence in our earlier case studies. However, our examples of high conceptual coherence are unconvincing due to the size of the ontologies and may simply be a structural function of a small and tightly interrelated ontology. We hope to find either that Powerpoint has a high conceptual coherence or that some of the 'tools' in Yahoo IM have a high coherence and that these examples possess a large enough set of concepts to be more convincing. Alternatively, we may find that none of these programs exhibit a high conceptual coherence which would still be acceptable due to their large size and late evolution. Nevertheless, we would like to have more data points to develop this metric further.

8.1.4 Potential Research Difficulties

We have anticipated these potential difficulties with the work and analysis:

- *Similarity of Applications* – While we suspect that MS Powerpoint has a different evolutionary history, its ontology may be too similar to that of MS Word due to its history of being sold as a suite of applications. There may be objections to researching a second Microsoft application. Alternative Reef-like applications that have the similar properties (size, evolutionary history) are Macromedia Freehand, Adobe Photoshop, and Quicken 2002.
- *Application size and complexity may hinder excavation.* – Each application has hidden or buried features. In the Microsoft products, both applications share packages that are installed from the Office suite and it may not be clear which parts of the ontology belong to the overall suite as opposed to the individual application. In Yahoo! IM, parts of the application invoke external web pages which blur the distinction between where the application boundaries sit. Also, some of Yahoo!'s features belong to very large domains. For example, sports scores

link to pages that describe the team or athlete in question. To save time and to avoid having to model the extensive domains underlying the simple system, we will probably need to make a distinction between concepts that are relevant to the use of the application and concepts which define the external domain that the application supports. For example, we will model the concepts of 'sport', 'team', and 'athlete' but not necessarily 'season' or 'field goal' as the latter concepts cannot be directly accessed through the Yahoo! IM application.

- *Ontologies may not map to proposed conceptual structures.* – While we expect to find that these applications match our proposed structures, it is also possible that they will not match or will lack the ontological elements that will allow us to draw these conclusions. This may be especially true of Reef and Urban structures where there may exist a fine line between an ontology possessing a large cluster of core concepts and an ontology that has multiple groups. It may also be possible that Word 2000 may indeed have multiple core concepts but that they cannot be identified using our methods.

8.2 Study 2: Develop use case silhouette for systems from Study 1.

In Study 1, we will have excavated ontologies from three systems and made some claims about the importance of the concepts within those ontologies. In Study 2, using a set of independently derived use cases, we will develop use case silhouettes for the excavated ontologies. We have chosen to obtain use cases from books describing how to use the applications from Study 1. To reduce the variability associated with authorship and approach, we have selected all the books from the *for Dummies* series in the hope that the style and series editing have ensured continuity across them. The books are:

- *Word 2000 for Dummies* by Dan Gookin
- *Powerpoint 2000 for Dummies* by Doug Lowe
- *Yahoo! for Dummies* by Brad Hill

Each book is broken down into chapters that describe how to use specific functions. In our case studies (see Appendices), we used help files and instruction sheets as our source of use cases. Instructions from these sources usually describe the sequence of morphological elements needed to activate a particular feature of an application. Another potential source of use cases would naturally be human experts familiar with these systems. We intentionally avoided choosing this route to reduce the variability in our analysis. These books already represent a snapshot of expert knowledge. By using a static source, we reduce both the likelihood that we will miss use cases (which is very likely if we were simply conducting interviews) and the likelihood that these use cases have been described incorrectly. This also provides an independent source that can be used to review our methodology and data.

From this analysis, we will answer the following questions:

- Do the silhouettes show ontological coverage that parallels the core and peripheral concepts identified by structural metrics? Are there peripheral concepts that appear in the same number of use cases or more as core concepts?
- Do the silhouettes show coverage corresponding to the teleons identified by structural metrics?

8.2.1 Validating Ontological Coverage

In our previous work, we hypothesized that core concepts have low sensitivity to modeling errors because they tend to possess many relationships and dependencies to other concepts [99]. Likewise, for use case silhouettes, we hypothesize that core concepts will be invoked more often than peripheral ones in a set of use cases, provided that the set represents a significant percentage of the total number of potential use cases that can be associated with a particular computing application. We are also interested in finding concepts that were declared to be peripheral by our mathematical methods but which appear frequently across the use cases. Such concepts may point to differences between theoretical models of system usage and models of actual usage.

8.2.2 Use Cases and Conceptual Coverage

We expect that some use cases will have a higher conceptual coverage than others, suggesting that they may be major use cases for that application or have a high complexity. We also believe that use cases that require a disproportionate number of core concepts will be those that may be used most frequently in actual use.

8.2.3 Potential Research Difficulties

We have anticipated these potential difficulties with the work and analysis:

- *Systematic errors in morphology / ontology linking* – Use case silhouettes assume that an application's morphology invokes the underlying ontology. Therefore each morphological element reveals one or more concepts in the ontology. If a frequently-used morphological element across all the use cases has not been correctly linked to the concepts it reveals then this will produce errors in the analysis.
- *Analysis sensitivity to use case selection* – Use case silhouettes are sensitive to the set of use cases chosen and how those use cases have been described. As articulated above, a use case may be extensive because the task itself is complex. Thus the number of concepts expressed in a single use case or even a subgroup of the use cases may skew the overall analysis. Likewise, the books we have chosen may have chapters dealing with difficult or non-obvious features delivered by the application that will produce more use cases than chapters dealing with ordinary usage.
- *Fidelity with respect to actual usage* – Use case silhouettes give relatively equal weight to each unique use case. They do not account with the frequency with which the use case is actually invoked during actual use. This may produce an analysis that inaccurately reflects the importance of a particular concept.
- *Difficulty of the work* – We expect this study to be the most labor intensive of the three studies because of the number of potential use cases in the books.

8.3 Study 3: Map usability data to a system

The excavation and analysis of ontologies in COTS applications has thus far been isolated from issues of actual human users and usage. Since 'bloat' and 'feature creep' are terms that depend on perception, it is necessary that we tie our ideas of conceptual coherence and ontological coverage back to actual use. In Study 3, we have obtained usability data that Dr. Joanna McGrenere gathered for her PhD thesis [145]. McGrenere surveyed a pool of Word 2000 users using screen shots and questionnaires and had them evaluate which 'functions' they recognized or were likely to use in their everyday work. She selected the 265 functions from the first-level functions on the default interface. We intend to map this data to our excavated ontologies to answer the following questions:

- Is there a correlation between the functions that users purport to need and the core concepts of a system?
- What is the coverage of the used functions relative to the overall ontology?
- How useful are these ontological representations for answering questions about 'bloat' and usefulness?

8.3.1 Usage centrality and ontological centrality

In theory, because core concepts represent those concepts essential to a computing application, it seems reasonable to believe that their appearance in actual practice would be unavoidable. Therefore, we expect to find a correlation between the functions actually used and the core concepts of a system.

8.3.2 Conceptual coherence and actual usage

If 'bloat' and 'feature creep' are perceptions that an application has delivered more functions than would otherwise be ordinarily used then we expect the user data for Word 2000 to reveal that only a fraction of the total number of features are known and utilized. As McGrenere's population gave answers in her questionnaire that suggested that they found Microsoft Word 2000 to be 'bloated', we would expect that those functions that they declared to be frequently used will produce a small ontological coverage relative to the whole ontology. We recognize that this is a partial 'straw man' argument because computing applications may have Zipf's Law characteristics where 10% of the features are invoked 90% of the time [89]. We wish to learn the actual percentage of concepts used and the percentage of core and peripheral clusters covered by used concepts. Lastly, by combining this user data and our own analysis, we would like to be able to provide another perspective on McGrenere's adaptable interfaces which offered users a pared-down morphology for Word 2000.

8.3.3 Applying ontological data to usability studies

In future work, the findings from our research will have to be correlated to domain models taken from actual use and user data. Usefulness is also a combined measure of both relevance and efficiency. To address the latter point, our work will have to integrate data from previous and current studies on usability. We would like this study to show that data from a usability study can be used in conjunction with our ontological data to further our understanding of software design and its relationship to usage.

8.3.4 Potential Research Difficulties

We have anticipated these potential difficulties with the work and analysis:

- *Data sensitivity to selection of user population* – While McGrenere's study covered a diverse population of users, it was not large enough to be considered a statistically significant sample of Microsoft Word users – something which would have been expensive to obtain and study. Thus, it may be that only a subset of the core concepts will be mentioned by the users or that some peripheral concepts will be found to be more prominent in McGrenere's user data. Different user populations with varying expertise and objectives will produce varying usage profiles. For example, one would expect a population of academic researchers to have different weights on certain word processing concepts than a population of lawyers. We will have to reexamine her user population during the analysis.
- *Mapping qualitative data to analysis* – McGrenere's data takes the form of questionnaire data which asks users to rate something as "familiar and unfamiliar" then "used regularly, used irregularly, and not used". We may need to conduct two separate analyses on this data: one for familiarity and one for frequency of use. It is not currently clear whether these measures should be combined to produce a more general valuation for mapping to the ontology.
- *Service coverage* – McGrenere identified 265 'functions' that she used in her study. Since these functions were selected from the first-level functions on the default interface they do not cover the entire set of morphological elements. Thus, it is very likely that the concepts invoked by these functions will not cover the set of concepts that were identified in Study 1. However, it may be possible that these functions do provide a sufficiently large set to allow an in-depth analysis.
- *Lack of similar data for other applications* – For the sake of completeness, this study should also include similar analyses from studies of these other applications. However, McGrenere's work represents the most thorough analysis of Word 2000. We have been unable to find similar studies for Powerpoint or Yahoo! IM. Lacking this detail, it may seem reasonable to expect that we would conduct our own user study but we believe that this is best left to future work.

9 Background Work

The following sections outline the ideas and technologies that contributed to this work.

9.1 Usefulness and Usability

We have proposed that conceptual coherence can be used to measure the usefulness of a computing application. Software developers have approached this problem from the perspective of improving software quality through testing activities and formal methods, user interface design and usability engineering, applying empirical methods to requirements development, and end-user evaluations to measure perceived usefulness.

9.1.1 Software Quality

Software developers tend to focus on the engineering aspects of system development taking the perspective that usefulness can be ensured by building what the customer requests. These requests are collected by requirements engineers and refined into requirements specifications [53, 106]. Software testing activities measures the conformity of the system design and correctness of implementation to these specifications [24]. Specifically, they measure the precision and correctness with which the system adheres to the customer's requirements using verification and validation testing activities throughout the development process. Verification activities test whether the product conforms to the specifications derived from the customer requirements and validation activities test whether developers are "building the right product" [177]. Theoretically, validation activities should ensure usefulness for the customer. However, Sommerville argues that validation activities cannot be performed on requirements specifications due to the lack of a frame of reference.

> *"The main problem of requirements validation is that there is nothing against which the system can be validated. A design or a program may be validated using the specification. However, there is no way to demonstrate that a requirements specification is correct. The validation process can only increase your confidence that the specification represents a system which will meet the real needs of the system customer." [187]*

Because of this lack of a reliable framework, the validation of computing applications has been limited to systems such as embedded systems that can be tested using formal methods and quantifiable testing techniques [2, 97, 209]. Formal methods can produce formal specifications that can be validated by inspection, assuming that the requirements are clear and the specifications are well organized [107]. During the requirements process, a formal method can clarify the informal statements made by customers [206] or provide reasoning techniques to identify inconsistencies or gaps in the specifications [33]. However, these formal methods assume clarity, consistency, and domain understanding from the customers and are therefore primarily verification and not validation activities. Thus, while software engineering techniques exist that can test whether the developing system conforms to the software's "blueprint", the requirements specifications, no techniques exist to determine if there are fundamental gaps or errors in the specifications themselves.

9.1.2 User Interface Design and Usability Engineering

The other software development activities that attempts to ensure usefulness are user interface design and usability testing [60, 95]. Usability engineering focuses on the correspondence between the user interfaces of a computing system and the user's conceptual models of how the

tasks should be performed. The techniques used by usability engineers include task analysis, user testing, iterative design, participatory design, and prototyping [157, 158]. However, these activities simply seek to ensure that the user can access the features of the software as efficiently as possible and that the external presentation of the software matches the user's understanding of these features. Again, these are engineering concerns and assume that the functionality of the software has been precisely articulated by the end users of the system.

9.1.3 Empirical Methods and Requirements Gathering

Researchers and developers have recognized that the requirements form the templates used to design and implement the eventual system have to be written from a thorough understanding of the user's domain and that this understanding can only be gained from interactions with the actual users in their working environment [50, 163, 173, 176, 186]. A number of empirical techniques have been developed to derive user requirements directly from the use context. These include incorporating ethnographic methods [77, 102], contextual design [26, 27, 96], intent-based specifications [136], and inquiry-based analysis [175]. In the human-machine systems area, ecological interface design and ecological task analysis use empirical studies of the work domain to improve the design of user interfaces [68, 122, 201, 202].

9.1.4 End-User Analysis

Information systems researchers consider usefulness in the context of *perceived usefulness* defined as "the degree to which a person believes that a particular system would enhance his or her job performance" and *perceived ease of use*, which he defines as "the degree to which a person believes that using a particular system would be free of effort" [1, 54]. Techniques for assessing the usefulness of a particular system consist of end-user interviews and surveys [1, 54, 55, 73, 119, 127].

9.2 Software Evolution

Software evolution research began with the seminal work of Lehman and Belady who studied the evolution of the IBM OS360 operating system. They tracked size of the system over time by the number of modules and found that it became increasingly unstable as developers added functionality to the basic system as part of normal maintenance activities. From this, Lehman derived his Laws of Software Evolution [128]. Since then, there have been many similar studies of software evolution, tracking changes to system elements such as source code, number of modules, and overall stability [9, 19, 41, 75, 78, 121]. There have also been many studies showing how software should or could be evolved to achieve goals such as greater stability, fewer errors, and improved maintainability [12, 13, 20, 45, 52, 112, 113, 138, 151, 164, 165]. While this research has contributed to our understanding of software's internal composition as it is adapted over time, we still know very little about how enhancements to software affect the users of the system.

There have only been two studies that have attempted to track the evolution of software by its feature enhancements: a study of telephony conducted by Antón and Potts [7, 8] and a study of Microsoft Word by Hsi and Potts [98]. A study by Godfrey on the evolution of the Linux system did track growth by the major subsystems but studied this through lines of code rather than changes to functionality or ontology [75]. Lehman is currently developing a theory of software evolution that accounts for feature evolution through feedback loops in the global software process [131-133].

9.3 Features and Services

The word "feature" is a useful term for referring to an application's services. Czarnecki and Eisnecker point out that features are a natural way to express concepts because they correspond to "chunks" used in human memory [52]. Chunking, or clustering, groups concepts and make them easier to remember [44].

9.3.1 What is a Feature?

"Features" have different meanings depending on the perspective or stage of software development that its used. At the requirements stage, features are clustering of individual requirements that describe a "cohesive, identifiable unit of functionality." [196, 197] or part of a specification that "a user perceives as having a self-contained functional role." [85]. Developers view features as simpler units of functionality [142]. For example, Cusumano and Selby – in describing Microsoft's culture – say the following:

> *"The features in Microsoft products are relatively independent units of functionality visible to end users. They are like building blocks, especially for applications products. Examples are printing, automatically selecting a column of numbers and adding them, or providing an interface to a particular vendor's hardware device. Features in systems products, such as Windows NT or Windows 95, are often less visible to the end user; Microsoft and other companies sometimes simply call these 'functions'." [51]*

In system development, features are sometimes perceived as "packages of incrementally added functionality", describing how feature enhancements are added in stages to a system.[34, 42, 43].

9.3.2 Feature-based Engineering techniques

Several software development techniques use features as their unit of development. These include feature engineering [196, 197], Feature-Oriented Domain Analysis (FODA) [79, 117], Feature-Oriented Reuse Method (FORM) [118], Feature Oriented Programming (FOP) [22, 23], and generative programming [52]. The general structure of these methods is to identify features in the problem domain, refine the concepts expressed by these features, and develop the supporting design and architectures around these features.

Product lines and product families are another software engineering method that uses features as an organizing principle for development. One paper defines a product line as having a reusable infrastructure of shared behaviors and services and allows the construction of many family members [59]. Another paper defines product families as "sets of products that share architectural properties, features, code, components, middleware, or requirements. [125]" While these terms seem to be interchangeable or depend on granularity,
features are used in both cases to develop a shared infrastructure for reuse [125, 139, 194].

9.3.3 Function Point Analysis

Function point analysis is a metric that measures the size of systems based on those system attributes perceivable by users. It abstracts all features into five types of components: external inputs, outputs, inquiries, external interfaces to other systems, and the logical internal files. These components can be further classified by their complexity [195]. Function points have been shown to be useful for effort and cost estimation [114, 120]. However, these metrics have also been criticized for oversimplification and relevance to technology [111].

9.4 Reverse Engineering and Program Understanding

9.4.1 Program Comprehension and Reverse Engineering

Our method for ontological excavation is an activity of program comprehension – "the process of acquiring knowledge about a computer program [181]." Specifically, our excavation methods are a type of reverse engineering defined by Chikofsky and Cross as "the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction. [47]" Rugaber describes five gaps that complicate the conceptual understanding of programs [181]:

- The gap between a problem from some application domain and a solution to it in some programming language.
- The gap between the concrete world of physical machines and computer programs and the abstract world of high-level descriptions.
- The gap between the desired coherent and highly structured description of a system as originally envisioned by its designers and the actual system whose structure may have disintegrated over time.
- The gap between the hierarchical world of programs and the associational nature of human cognition.
- The gap between the bottom-up analysis of source code and the top-down synthesis of the description of the application.

Ontological excavation addresses the first three of these gaps but only indirectly as we use black box reverse engineering methods to reverse engineer the domain of the system as opposed to white box methods that directly examine the system's implementation.

9.4.2 Black Box Reverse Engineering

Black box reverse engineering follows the tradition of other qualitative research methods such as those in ecological psychology, ethnography, and cognitive anthropology. These methods are designed to develop an understanding of a subject's domain by analyzing their use of language and artifacts in the subject's environment. These contextual or naturalistic approaches, as opposed to abstractionist or laboratory approaches, take the perspective that human behavior is grounded in an environment and context and that they can only be properly understood in that context [103, 108, 155, 156]. These methods also stress that their application be performed with the absence of preconceived notions about what will be discovered [137, 189]. Our black box reverse engineering techniques are grounded in this tradition. By treating the computing application as the unit of analysis, and referencing domain knowledge directly from the application whenever possible, we can construct the ontology as encoded in the application.

9.4.3 Domain Analysis and Reverse Engineering

Understanding the domain descriptions from either a computing application or from work products developed through user interactions seem to be an important prerequisite to building (or reverse-engineering) a useful computing application. Previous work in domain analysis and reverse engineering has developed methods for extracting the domain from program documentation [5], requirements specifications [77], code [48, 58], and interviews with domain experts [10]. Of these techniques, code domain analysis might be the best method for automating

ontological excavation but code itself contains a meta-domain with concepts and relationships that concern software engineering and programming.

9.5 Interface Models and Recovery

9.5.1 User Interface Representations

We adopted our morphological elements taxonomy from the Swing component framework in Java [192]. Besides the standard graphical user interface (GUI) methods [60, 95], there are a variety of alternative user interface representation techniques that include state transition networks, application frameworks, and context-free grammars [152, 154].

9.5.2 Automated Recovery of User Interfaces

There have been a number of approaches for reverse-engineering user interface models for the purposes of reuse and testing. MORPH, the Model-Oriented Reengineering Process of Human-Computer Interfaces, uses static code analysis and recovers interface designs from character-oriented user interface designs and transforms them to graphical user interfaces [153] using an abstraction hierarchy based on Foley's basic interaction techniques [69]. CelLEST reverse engineers legacy interfaces based on user interactions and the construction of a GUI or a web interface from these interactions [190, 191]. GUITAR is a system for testing GUIs that automatically generates a GUI representation through interaction with the application. The GUI is represented as a graph with nodes consisting of window states. Windows are modeled as a set of widgets (e.g. buttons, labels, text fields) that comprise the window, its properties, and the values associated with those properties. It also distinguishes between modal and modeless windows [150].

9.6 Ontologies

Traditionally, ontology is "a branch of philosophy dealing with the *a priori* nature of reality" [39, 40, 82]. In computer science, the word ontology is used to describe a set of concepts or representation of these concepts for domain and data modeling. However, the grounding provided by the philosophical formalisms have been used to refine and concretize data modeling formalisms [82-84, 204]. Ontologies in computer science have been designed for representing knowledge in intelligent systems [21] and exchanging data between knowledge databases used in applications such as web searches and e-commerce [32, 35, 80, 81, 135, 148, 149].

9.7 Graph Analysis Tools

9.7.1 Centrality Metrics

Our semantic network is a graph – a mathematical structure consisting of nodes that are connected by edges. Graphs are well understood structures in both mathematics [90, 91] and computer science[49, 184]. The set of graph tools that we adopted for our work come primarily from social network theory although similar algorithms have been used to analyze the design and evolution of architectural structures and cities. Hillier used graph algorithms on the intersections of street maps to derive the transit patterns of its inhabitants [93, 94]. Social Network theorists use graph measurements that they call *prestige measures*. Prestige measures assess the importance of a node relative to the rest of a graph. For example, one mathematically simple measure counts the number of edges incident to a node (or in- and out-degrees in a directed graph). This *degree centrality* may find a person in a social network who is important because they possess many connections to other individuals. These techniques have been used to study

situations like the relationships of the Medici families and their marriages to other families during the Italian Renaissance [205].

9.7.2 Cluster Analysis

We used a *k-core analysis* to identify teleons in our ontologies. This method belong to a class of techniques called *data clustering* that are used to identify significant subgroups in a dataset or graph [86, 110]. Besides those applications to social network theory [30], these techniques have been applied to information recognition [110], Web topologies [101], and the reverse engineering of objects from legacy code [198].

9.8 Use Case Silhouettes

The idea for use case silhouetting came from reading studies in cognitive neuroscience which examined the electroencephalograms [71], X-Ray computed tomography, or Magnetic Resonance Images (MRI) [178] of subjects performing certain cognitive tasks to identify which regions of the brain became active during this process. Since we were interested in identifying those concepts which became active during a task, it seemed reasonable to adopt a similar approach to visualizing the active portions of an application's ontology. In cognitive neuropsychology and cognitive neuroscience, subjects are given tasks to perform designed to invoke cognitive behaviors such as visualization or memory retention and recall [66]. Since we wished to identify concepts which might be invoked during actual usage, it seemed reasonable to turn to an activity based model for our data and to use the application morphology as the input into the system.

Use Cases are part of the Unified Software Process (USP) [109] and describe "a sequence of actions, including variants, that a system performs to yield an observable result of value to an actor. [29]" They are related to a class of techniques in requirements engineering called scenario-based requirements engineering [116, 174, 193]. Scenarios describe a sample procedure or execution of a system by presenting specific, concrete episodes. These narratives, gathered from users [53, 175, 176], can then aid in requirements gathering tasks such as goal and obstacle identification.

Our use case silhouette methodology has similarities to user interface analysis techniques such as GOMS (Goals, Operators, Methods, Selection) [44], task analysis [60], run-time behavior and system logs to track user behaviors [62, 63], sequence models [27], and cognitive walkthroughs [60, 157]. The general objective of these techniques is to identify or verify a sequence of actions that the user will use to perform a task that will contribute to the completion of a goal. In software engineering, our technique is strongly related to El-Ramly and Stroulia's work on using system-level traces of user interactions to develop requirements [62, 63]. Use case silhouetting relies on tracing a sequence of activities specified by a use case to determine which concepts are invoked in the ontology. Certainly, in future work, use case silhouetting could be combined with usability testing techniques, such as those using user interface events [92], to apply ontological coverage as a metric for measuring usefulness or usability.

10 Expected Research Contributions

Our main contribution will be to demonstrate that a property which we call *conceptual coherence*, can be measured and analyzed from the ontology of a computing application, that it can be tied directly to perceptions of usefulness, and that it is negatively affected by the presence of too many features that contain peripheral concepts.

10.1 Summary of Major Contributions

- A theory of software ontology and *conceptual coherence* as an aspect of *conceptual integrity*.
- Methodology for *ontological excavation* – the black box reverse engineering of a software ontology – enables us to determine those concepts encoded into the system that are visible and accessible to users of the system.
- Domain-independent analysis methods for identifying the *core concepts* of an application – those concepts that are *essential* to the definition and function of that application.
- A theory of software features and their expression at the ontological level in the form of *teleons*.
- Domain-independent analysis methods for identifying a teleon in an ontology.
- Methodology for measuring the ontological coverage of a set of use cases using *use case silhouetting*.
- A method for identifying features for an application allowing us to study the synchronic variation of these features across applications and their diachronic variation across application versions.

10.2 Summary of Minor Contributions

- A model of the user interface that we call a morphological map that can be used to measure the activation cost of a concept.
- A software ontology model that can be enhanced and adapted to more robust models for design.
- A supergraph that shows the relationships of morphological elements to concepts.
- A bipartite graph allowing analysis of the relationship between morphological elements and its concepts.
- Graph metrics obtained from social network theory that can be used to identify critical concepts in an ontology or software architecture.
- Three case studies of well-known applications that will produce ontologies and morphological maps for each of them.
- A method for mapping usability data to the morphology and ontology of an application as an approximation of conceptual fitness.

10.3 Other Contributions

These are potential applications of our research findings that we will pursue further in the dissertation research.

- Method for assessing the potential success of a software system prior to delivery to a set of customers.

- Techniques for component-based design around a central system that implements only the core concepts required by a specific use context. The resulting architecture may be more amenable to dynamic adaptation based on user demand.
- An alternate design methodology for user interfaces using the ontology and use case silhouetting as a guideline for assigning elements to minimize activation cost to user.
- Method for measuring the potential usability of a system from the relationship of an application's morphology to its ontology.
- Recommendations for developers for identifying core features of system and to measure conceptual coherence for purposes of feature-set control. [143]
- Method for experimental modifications to a software's ontology to increase or reduce prominence of various concepts to determine potential impact of that conceptual model to the overall system. This can also be applied to the addition (*ontological grafting*) or subtraction (*ontological pruning*) of new concepts.
- A taxonomy of features and computing applications that can be used to provide a clearer categorization scheme for computing applications and a development resource for designing them.

11 Plan of Completion

There are two plans here. The first conforms, more or less, to the studies outlined in Section 8. The second is optimized towards obtaining publishable results for upcoming conferences.

11.1 Basic Schedule

Task description	Estimated Time
Study 1: Exploratory study on large systems	
Excavate ontology of Powerpoint	2 weeks
Analyze Powerpoint ontology	2 days
Excavate Ontology of Yahoo Messenger	3 weeks
Analyze Yahoo Messenger ontology	2 days
Excavate ontology of Word	3 weeks
Analyze Word ontology	2 days
Study 2: Develop use case silhouettes	
Develop use case silhouette of Powerpoint	1 week
Develop use case silhouette of Yahoo Messenger	1 week
Develop use case silhouette of Word	1 week
Analyze use case silhouettes against respective ontology analyses	1 day
Study 3: Map usability data to system	
Map McGrenere's data to morphology and ontology of Word	2 days
Analyze data for results	1 day

11.2 Optimized Schedule for Publishing

Task description	Estimated Time
Excavate ontology of Word	3 weeks
Analyze Word ontology	2 days
Develop use case silhouette of Word	1 week
Map McGrenere's data to morphology and ontology of Word	2 days
Analyze data for results	1 day
Excavate ontology of Powerpoint	2 weeks
Analyze Powerpoint ontology	2 days
Develop use case silhouette of Powerpoint	1 week
Excavate Ontology of Yahoo Messenger	3 weeks
Analyze Yahoo Messenger ontology	2 days
Develop use case silhouette of Yahoo Messenger	1 week
Analyze use case silhouettes against respective ontology analyses	1 day

11.3 Projected Chapters in dissertation to be written

- Background Work – Will likely require enhancements to current papers and descriptions in the proposal.
- *Methodology Section: Surveying the Morphology* – In depth descriptions of the methodologies and heuristics used to develop the morphological map.
- *Methodology Section: Excavating the Ontology* – In depth descriptions of the black box and conceptual modeling techniques used to identify and model concepts and relationships from the morphological map.
- *Methodology Section: Ontological Analysis* – In depth descriptions of the analysis techniques used to study the ontology.
- *Methodology Section: Use Case Silhouetting* – In depth description of the use case silhouette methodology.
- *Threats to Validity in Ontological Excavation and Use Case Silhouetting* – Describes the sources of variability in the methods used in this study. Where appropriate, includes data from experimental studies, performed prior to proposal, that test the rigor of the analysis.
- *Feature Growth and Morphological Complexity* – Describes relationship of feature evolution and the morphological complexity of the system.
- *Computing Ecosystems and Use Niches* – Describes use contexts in the context of computing ecosystems and their individual use niches to motivate the biological metaphor of fitness.
- *Future Work*

12 Glossary

Underlined words denote terms that were coined or re-defined for this specific research.

activation cost	The amount of effort required by a user to access a service provided the application
actor	A type of user of a computing system. [109]
adaptation	The process of changing attributes and behaviors of something to better suit a specific context. In biology, it also describes a physiological attribute of a species that improves its fitness relative to an element of the surrounding ecosystem. [56, 57]
aggregation	A whole/part relationship where one class of entity types represents a larger thing which consists of smaller things. Denoted by a 'has-a' relationship. [29]. In our modeling conventions, we break the traditional convention of requiring both things to have independent identities in the case of attributes. However, attributes themselves are not permitted to have has-a relationships.
association	A structural relationship that specifies that elements of one type are connected to elements (concepts) of another type. [29]
attribute	An intrinsic property of a thing in the real world. [203] In our model of an application ontology, an attribute lacks independent existence except as a property of an entity type.
betweenness centrality	A prestige measure that measures the number of geodesics between all pairs of nodes in the graph that use a particular node. The higher the centrality measure, the more other nodes depend on that node. Because leaf nodes only serve as start and end points for paths, they automatically have a betweenness value of 0. [205]
bipartite graph	A graph in which the nodes can be partitioned into two subsets such that edges always connected nodes taken from the different subsets. Bipartite graphs are used to model two-mode networks. [205]
black box reverse engineering	The recovery of some computing application domain model, behavior, or attribute without reference to the code used to implement that computing application.
bloat	The term used to describe a computing application possessing a disproportionate number of unnecessary services that interfere with the normal or desired use of this application.
closeness centrality	A prestige measure that measures the average distance from a subject node to all other nodes. [205]
computing application	Any device or system that uses some form of computation to accomplish a goal. Also the term that can refer to 'application', 'computing artifact', 'software', 'software application', and 'software system'.

concept	A generalized idea of a thing or class of things. [185] In our model of an application ontology, either an entity type or an attribute can be a concept.
conceptual coherence	A property of a computing application measuring the degree to which the concepts contained within its ontology are tightly related.
conceptual fitness	The property of a computing application that assesses how well its ontology matches the domain of the use context in which it is being used.
conceptual integrity	The property of a system designed under a unified and coordinated set of design ideas. [37]
container	A morphological element that contains and structures interactors [99]
core concept	A concept that is essential to defining a computing application's feature set and identity. [99]
correction	A software maintenance activity that applies repairs to errors in the code [166]
customer	The purchaser of the computing application. Not necessarily the user of the application.
degree centrality	A prestige measure that uses the number of edges on a node (its degree). A value of 1.0 on a scale of 0.0 to 1.0 means the node has edges leading to all other nodes in the graph. [205]
density	The number of edges in a graph divided by the possible number of edges. Also called network density. [31]
diachronic variation	Variation across time – usually in reference to evolution or development.
digraph	A directed graph. [205]
display	A morphological element that makes both static and dynamic data about the computing application's states available to the user. [99]
domain model	"A definition of the entities, operations, events, and relationships that abstract commonalities or regularities in a domain, together with a classification of these." [10]
ecosystem	"An ecosystem is a system of interacting species in a particular environment." [123]. Defines a system of interest where the granularity could be the object of study (like a species) or a set of arbitrary conditions. [140]
eigenvector centrality	A prestige measure that measures the centrality of a node relative to the importance of its surrounding nodes. [205]
enhancement	A software maintenance activity that adds new features, generally visible to the users of the system. [166]
entity	A "thing" that can be distinctly identified. [46]
entity type	A set of entities that have the same attributes. [61]

E-type program	A software system that solves a problem or implements a computer application in the real world. [129]
evolution	The process of change over a period of time. In the biological sense, evolution refers to the physiological changes that a species experiences through the process of mutation, natural selection, and reproduction. [56, 57]
feature	A user-accessible behavior or service implemented by a computing application.
feature aggregation	An evolutionary behavior of a computing application where it acquires new features at every stage of release.
feature creep or creeping featurism	The "tendency to add to the number of features that a device can do, often extending the number beyond all reason." [162]
fitness	Attribute of an entity that assesses its ability to inhabit a specific context. In biology, fitness describes the ability of an organism or a species to survive long enough to reproduce.
generalization	A relationship between a kind of entity type (parent or superclass) and a more specific kind of that entity type (type, child, or subclass). Denoted by an "is-a" relationship. [29]
geodesic	The shortest path between a pair of nodes. [205]
improvement	A software maintenance activity that applies an optimization to performance, usability, maintenance, or other nonfunctional properties of a computing application. [166]
instance	A concrete manifestation of an entity type [29].
information centrality	A prestige measure that measures the information contained in all paths originating with a specific node. [205]
interactor	A morphological element that can be directly accessed or manipulated by the user of a system. [99]
k-core	A connected, maximal, induced subgraph of nodes such that each node has a minimum degree greater than equal to k [65].
morphological element	A component that forms the structure of a computing application's morphology.
morphological map	A graph modeling the elements that compose the morphology of a computing application and their relationships to each other. [99]
morphology	The external presentation of a computing application consisting of those elements that are both user accessible and percievable.
niche	A place and functions that a species has in an ecosystem
ontological coverage	A metric that measures the proportion of the ontology covered by a set of concepts.
ontological	The process of using black box reverse engineering to recover a computing

excavation	application's ontology. [99]
ontological grafting	The process of adding concepts or a set of concepts and their relationships to an ontology.
ontological pruning	The process of removing concepts or a set of concepts and their relationships from an ontology.
ontological structure	A structural pattern in the ontology that organizes the concepts and their relationships.
ontology	A representation of set of concepts used for domain or data modeling. [35, 67, 81, 147, 203, 204]. Also the study of being – of existence and its relationship to nonexistence [123].
operations	The activities that a system performs.
perceived ease of use	The degree to which a person believes that using a particular system is free of effort [54]
perceived usefulness	The degree to which a person believes that a particular system could enhance his or her job performance [54]
peripheral concept	A concept which is considered optional to an application's definition. [99]
portal	A mapping from the morphology of a computing application to a concept or set of concepts in the ontology. [98]
prestige measure	A prestige measure assesses the importance of a node relative to the rest of a graph. [205]
problem domain	A collection of items of real-world information that have the following characteristics: 1) "deep or comprehensive relationships among the items of information are suspected or postulated with respect to some class of problems" and 2) the problems are perceived as significant by the members of the community. [10]
problem frame	A diagram that describes the class, characteristics of the problem domain, and a central concern for a class of problems. [105]
reef ontological structure	An ontological structure with a core that exists not only to support itself but also a number of other entity types that contribute to the overall system.
relationship	A reference or association that exists between entity types. [61]
requirement	A description of how a system should behave or a description of a system property or attribute. [187]
semantic network	The collection of all the relationships that concepts have to other concepts [188]. Semantic networks are the first ontology models to make use of graphical formalism and were developed as psychological models of human memory [18] [185]. Generically speaking, they are graphical representations of a body of facts [160].
service	A service is an operation or series of operations performed by an

	application that performs a task for a user.
social network	A graph or network that encapsulates people or social groups and their relationships to one another. [205]
software evolution	The process of adapting a computing application or system during the software maintenance phase of its development. Also the description of the changes that software experiences over its lifetime.
software product family	"A set of products that share architectural properties, features, code, components, middleware, or requirements." [139]
software product line	"A set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way." [115]
subtype	A *subtype* is a specialization of an entity type [29].
superpositioned graph (supergraph)	A graph containing a computing application's morphological map, the ontology, and the interconnections that link a morphological element (portal) to the concepts that it reveals. Used to derive the bipartite graph of morphological elements and concepts.
synchronic variation	The variation of features across different entities of the same type within the same time frame.
teleon	An identifiable substructure of an ontology that suggests features at the user level. Consists of a set of concepts that have strong interrelationships. [98]
toolbox ontological structure	A Toolbox structure has a collection of conceptually unrelated and lightly related ontologies that have been assembled for reasons of convenience or design under a single morphology.
urban ontological structure	An ontological structure with multiple clusters of core concepts that are loosely connected to each other.
usability	An attribute of an application that measures how much effort is required to activate an affordance or service provided by that application.
use case	"A use case specifies a sequence of actions, including variants, that the system can perform and that yields an observable result of value to a particular actor." [109]
use case coverage	A metric for the proportion of concepts in an ontology covered by a set of use cases.
use case silhouette	The set of concepts that have been activated or illuminated by a set of use cases or a sequence of morphological element activations.
use context	The external physical (or virtual) environment that contains a computing application and its users, the goals that the combined computing application/user system wishes to achieve, and the various nuances (business rules, customer demand, user and system capabilities) that

	govern the operation and performance of both environment and goal completion.
use ontology	The use ontology consists of only those concepts that are actually used in a specific use context.
usefulness	The extent to which an application succeeds in assisting a set of users to achieve a set of goals, relative to the amount of effort required to engage those features
user	The person, group of people, or entity that uses a computing application.
workpieces problem	"A *problem* of developing a tool to support creation and editing of text or other machine-readable objects." [105]

13 Bibliography

[1] D. Adams, R. R. Nelson, and P. A. Todd, "Perceived Usefulness, Ease of Use, and Usage of Information Technology: A Replication," *MIS Quarterly*, pp. 227-247, 1992.

[2] W. R. Adrion, M. A. Branstad, and J. C. Cherniavsky, "Validation, Verification, and Testing of Computer Software," *ACM Computing Surveys*, vol. 14, pp. 159-192, 1982.

[3] C. Alexander, *The Timeless Way of Building*. New York, NY: Oxford University Press, 1979.

[4] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language: Towns, Buildings, Construction*. New York, NY: Oxford University Press, 1977.

[5] N. Anquetil, "Characterizing the informal knowledge contained in systems," presented at Eight Working Conference on Reverse Engineering, Stuttgart, Germany, 2001.

[6] A. I. Antón, "Goal-Based Requirements Analysis," presented at International Conference on Requirements Engineering, 1996.

[7] A. I. Antón and C. Potts, "Functional Paleontology: System Evolution as the User Sees It," presented at 23rd International Conference on Software Engineering, Toronto, Canada, 2001.

[8] A. I. Antón and C. Potts, "Requirements Engineering in the Long-Term: Fifty Years of Telephony Feature Evolution," presented at International Conference on Software Engineering, Toronto, ON, 2001.

[9] M. Aoyama, "Continuous and Discontinuous Software Evolution: Aspects of Software Evolution across Multiple Product Lines," presented at 4th International Workshop on Principles of Software Evolution, Vienna, Australia, 2001.

[10] G. Arango, "Domain Analysis Methods," in *Software Reusability*, W. Schafer, R. Priéto-Diaz, and M. Matsumoto, Eds. Chichester, England: Ellis Horwood, 1994, pp. 17-49.

[11] G. Arango and R. Prieto-Díaz, "Domain Analysis Concepts and Research directions," in *Domain Analysis and Software Systems Modeling*, R. Priéto-Diaz and G. Arango, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1991, pp. 9-26.

[12] L. J. Arthur, *Rapid Evolutionary Development*. New York, NY: John Wiley & Sons, 1992.

[13] L. J. Arthur, *Software Evolution: The Software Maintenance Challenge*. New York, NY: John Wiley and Sons, 1988.

[14] D. Attenborough, *The Living Planet: A Portrait of the Earth*. Boston, MA: Little, Brown and Company, 1985.

[15] R. Baecker, K. Booth, S. Jovicic, J. McGrenere, and G. Moore, "Reducing the Gap Between What Users Know and What They Need to Know," presented at ACM Conference on Universal Usability 2000, 2000.

[16] B. Balzer, "Living With COTS," presented at International Conference on Software Engineering, Orlando, FL, 2002.

[17] R. Barker, *CASE*Method: Entity-Relationship Modelling*. New York, NY: Addison-Wesley, 1990.

[18] A. Barr and E. A. Feigenbaum, "The Handbook of Artificial Intelligence," vol. 1. New York , NY: Addison-Wesley, 1989.

[19] E. Barry, S. Slaughter, and C. F. Kemerer, "An Empirical Analysis of Software Evolution Profiles and Outcomes," presented at 20th International Conference on Information Systems, Charlotte, North Carolina, 1999.

[20] D. Barstow and G. Arango, "Designing Software for Customization and Evolution," 1991.

[21] J. A. Bateman, "Ontology Construction and Natural Language," presented at Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation, PAdova, 1993.

[22] D. Batory, "A Tutorial on Feature Oriented Programming and Product Lines," presented at International Conference on Software Engineering, Portland, Oregon, 2003.

[23] D. Batory, C. Johnson, B. Macdonald, and D. Von Heeder, "Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study," *ACM Transactions on Software Engineering and Methodology*, vol. 11, pp. 191-214, 2002.

[24] B. Beizer, *Software Testing Techniques*, 2nd ed. New York, NY: Van Nostrand Reinhold, 1990.

[25] D. Benyon, T. Green, and D. Bental, *Conceptual Modeling for User Interface Development*. London: Springer-Verlag, 1999.

[26] H. Beyer and K. Holtzblatt, "Apprenticing with the Customer," *Communications of the ACM*, vol. 38, pp. 45-52, 1995.

[27] H. Beyer and K. Holtzblatt, *Contextual Design: Defining Customer-Centered Systems*. San Francisco: Morgan-Kaufmann Publishers, Inc., 1998.

[28] G. Booch, *Object-Oriented Analysis and Design*. Reading, MA: The Benjamin/Cummings Publishing Company, Inc., 1994.
[29] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley, 1999.
[30] S. P. Borgatti and M. G. Everett, "Models of Core/Periphery Structures," *Social Networks*, pp. 375-395, 1999.
[31] S. P. Borgatti, M. G. Everett, and L. C. Freeman, *UCINET 5.0 Version 1.00*: Analytic Technologies, 1999.
[32] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick, "CLASSIC: A Structural Data Model for Objects," presented at SIGMOD International Conference on Management of Data, Portland, Oregon, 1989.
[33] J. P. Bowen and M. G. Hinchey, "Seven More Myths of Formal Methods," *IEEE Software*, vol. 12, pp. 34-41, 1995.
[34] T. F. Bowen, F. S. Dworack, C. H. Chow, N. Griffeth, G. E. Herman, and Y.-J. Lin, "The Feature Interaction Problem in Telecommunications Systems," presented at Seventh International Conference on Software Engineering for Telecommunication Switching Systems, Bournemouth, UK, 1989.
[35] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Resnick, "Living with CLASSIC: When and How to Use a KL-ONE-Like Language," in *Principles of Semantic Networks*, J. Sowa, Ed.: Morgan Kaufmann Publishers, 1990.
[36] S. Brand, *How Buildings Learn*. New York, NY: Penguin Books, 1994.
[37] F. Brooks, *The Mythical Man-Month*. Reading, MA: Addison-Wesley, 1995.
[38] K. Brooks, "Dancing with digital interface complexity: a story approach," *IEEE Multimedia*, vol. 9, pp. 8-11, 2002.
[39] M. Bunge, *Ontology I: the Furniture of the World*, vol. 3. New York, NY: D. Reidel Publishing Co., Inc., 1977.
[40] M. Bunge, *Ontology II: A World of Systems*, vol. 4. New York, NY: D. Reidel Publishing Co., Inc., 1979.
[41] E. Burd, S. Bardley, and J. Davey, "Studying the process of software change: an analysis of software evolution," presented at Seventh Working Conference on Reverse Engineering, Brisbane, Qld. Australia, 2000.
[42] E. J. Cameron, N. Griffeth, Y.-J. Lin, M. E. Nilson, W. K. Schnure, and H. Velthuijsen, "Towards a Feature Interaction Benchmark for IN and Beyond," *IEEE Communications Magazine*, vol. 31, pp. 64-69, 1993.
[43] E. J. Cameron and H. Velthuijsen, "Feature Interactions in Telecommunications Systems," *IEEE Communications Magazine*, vol. 31, pp. 18-23, 1993.
[44] S. K. Card, T. P. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.
[45] E. Chang, E. Gautama, and T. S. Dillon, "Extended Activity Diagrams for Adaptive Workflow Modellling," presented at Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2001.
[46] P. P. Chen, "The Entity-Relationship Model - Toward a Unified View of Data," *ACM Transactions on Database Systems*, vol. 1, pp. 9-36, 1976.
[47] E. J. Chikofsky and J. H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, vol. 7, pp. 13-17, 1990.
[48] R. Clayton, S. Rugaber, and L. Wills, "Dowsing: A Tool Framework for Domain-Oriented Browsing of Software Artifacts," presented at 13th IEEE International Conference on Automated software Engineering, 1998.
[49] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridget, MA: MIT Press, 1994.
[50] B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, vol. 31, pp. 1268-1287, 1988.
[51] M. A. Cusumano and R. W. Selby, *Microsoft Secrets*. New York, NY: The Free Press, 1995.
[52] K. a. E. Czarnecki, Ulrich W., *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison-Wesley, 2000.
[53] A. M. Davis, *Software Requirements Analysis and Specification*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
[54] F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," *MIS Quarterly*, pp. 318-339, 1989.

[55] F. D. Davis, R. P. Bagozzi, and P. R. Warshaw, "User Acceptance of Computer Technology: A Comparison of Two Theoretical Models," *Management Science*, vol. 35, pp. 982-1003, 1989.
[56] R. Dawkins, *The Blind Watchmaker*. New York: W.W. Norton and Company, 1987.
[57] R. Dawkins, *The Selfish Gene*. New York: Oxford University Press, 1989.
[58] J. M. DeBaud, B. Moopen, and S. Rugaber, "Domain Analysis and Reverse Engineering," presented at International Conference on Software Maintenance, Victoria, British Columbia, Canada, 1994.
[59] J.-M. DeBaud and K. Schmid, "A Systematic Approach to Derive the Scope of Software Product Lines," presented at International Conference of Software Engineering, Los Angeles, CA, 1999.
[60] A. Dix, J. Finlay, G. Abowd, and R. Beale, *Human-Computer Interaction*. New York, NY: Prentice-Hall, 1993.
[61] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*. New York, NY: Addison-Wesley, 1994.
[62] M. El-Ramly, E. Stroulia, and P. Sorenson, "From Run-time Behavior to Usage Scenarios: An Interaction-Pattern Mining Approach," presented at 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, 2002.
[63] M. El-Ramly, E. Stroulia, and P. Sorenson, "Recovering Software Requirements from System-User Interaction Traces," presented at 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, 2002.
[64] D. W. Embley, B. D. Kurtz, and S. N. Woodfield, *Object-Oriented Systems Analysis: A Model-Driven Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1992.
[65] M. G. Everett and S. P. Borgatti, "Peripheries of Cohesive Subsets," *Social Networks*, pp. 397-407, 1999.
[66] M. W. Eysenck and M. T. Keane, *Cognitive Psychology: A Student's Handbook*. East Sussex, UK: Lawrence Erlbaum Associates Ltd., 1992.
[67] R. d. A. Falbo, G. Guizzardi, and K. C. Duarte, "An Ontological Approach to Domain Engineering," presented at International Conference on Software Engineering and Knowledge Engineering (SEKE'02), Ischia, Italy, 2002.
[68] J. M. Flach, "The Ecology of Human-Machine Systems I: Introduction," *Ecological Psychology*, vol. 2, pp. 191-205, 1990.
[69] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*, Second ed. Reading, MA: Addison-Wesley, 1990.
[70] M. Fowler, *Refactoring*. Reading, MA: Addsion-Wesley, 1999.
[71] W. J. Freeman, "The Physiology of Perception," *Scientific American*, vol. 264, pp. 78-85, 1991.
[72] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1994.
[73] D. Gefen and M. Keil, "The Impact of Developer Responsiveness on Perceptions of Usefulness and Ease of Use: An Extension of the Technology Acceptance Model," *The DATA BASE for Advances in Information Systems*, vol. 29, pp. 35-49, 1998.
[74] W. W. Gibbs, "Taking Computers to Task," in *Scientific American*, vol. 277, 1997, pp. 82-89.
[75] M. W. Godfrey and Q. Tu, "Evolution in Open Source Software: A Case Study," presented at International Conference on Software Maintenance, San Jose, CA, 2000.
[76] T. R. G. Green and D. R. Benyon, "The Skull Beneath The Skin: Entity-Relationship Models of Information Artefacts," *International Journal of Human-Computer Studies*, vol. 44, pp. 801-828, 1996.
[77] S. J. Greenspan, J. Mylopoulos, and A. Borgida, "Capturing More World Knowledge in the Requirements Specification," in *Domain Analysis and Software Systems Modeling*, R. Priéto-Diaz and G. Arango, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1991, pp. 53-62.
[78] R. M. Greenwood, B. Warboys, R. Harrison, and P. Henderson, "An Empirical Study of the Evolution of a Software System," presented at 13th IEEE Conference on Automated Software Engineering, Honolulu, HI, 1998.
[79] M. L. Griss, J. Favaro, and M. d'Alessandro, "Integrating Feature Modeling with the RSEB," presented at Fifth International Conference on Software Reuse, 1998.
[80] T. R. Gruber, "Ontolingua: A Mechanism to Support Portable Ontologies," Stanford University, Technical Report June 1992 1992.
[81] T. R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge sharing," in *Formal Ontology in Conceptual Analysis and Knowledge Representation*, N. Guarino and R. Poli, Eds.: Kluwer Academic Publishers, 1993.
[82] N. Guarino, "Formal Ontology, Conceptual Analysis, and Knowledge Representation," *International Journal of Human-Computer Studies*, vol. 43, pp. 625-640, 1995.

[83] N. Guarino and C. Welty, "Ontological Analysis of Taxonomic Relationships," presented at ER-2000: The 19th International Conference on Conceptual Modeling, USA, 2000.
[84] N. Guarino and C. Welty, "Towards a Methodology for Ontology-based Model Engineering," presented at ECOOP-2000 Workshop on Model Engineering, 2000.
[85] S. Guerra, M. Ryan, and A. Sernadas, "Feature-Oriented Specifications," School of Computer Science, University of Birmingham, Technical Report Nov 1996 1996.
[86] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, "On Clustering Validation Techniques," *Journal of Intelligent Information Systems*, vol. 17, pp. 107-145, 2001.
[87] T. Halpin, *Conceptual Schema and Relational Database Design*, 2nd ed. Sydney, AUS: Prentice Hall, 1995.
[88] T. Halpin, "Data modeling in UML and ORM revisited," presented at International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design, Heidelberg, Germany, 1999.
[89] S. J. Hanson, R. E. Kraut, and J. M. Farber, "Interface Design and Multivariate Analysis of UNIX Command Use," *ACM Transactions on Office Information Systems*, vol. 2, pp. 42-57, 1984.
[90] F. Harary, R. Z. Norman, and D. Cartwright, *Structural Models: An Introduction to the Theory of Directed Graphs*. New York, NY: John Wiley and Sons, 1965.
[91] F. Harary and E. M. Palmer, *Graphical Enumeration*. New York, NY: Academic Press, 1973.
[92] D. M. Hilbert and D. F. Redmiles, "Extracting Usability Information from User Interface Events," *ACM Computing Surveys*, vol. 32, pp. 384-421, 2000.
[93] B. Hillier, *Space is the Machine: A Configurational Theory of Architecture*. Cambridge, UK: Cambridge University Press, 1996.
[94] B. Hillier and J. Hanson, *The Social Logic of Space*. Cambridge, UK: Cambridge University Press, 1984.
[95] D. Hix and H. R. Hartson, *Developing User Interfaces*. New York, NY: John Wiley and Sons, 1993.
[96] K. Holtzblatt and H. Beyer, "Making Customer-Centered Design Work For Teams," *Communications of the ACM*, vol. 36, pp. 92-103, 1993.
[97] W. E. Howden, "Validation of Scientific Programs," *ACM Computing Surveys*, vol. 14, pp. 193-227, 1982.
[98] I. Hsi and C. Potts, "Studying the Evolution and Enhancement of Software Features," presented at Intl. Conf. Software Maintenance, San Jose, CA, 2000.
[99] I. Hsi, C. Potts, and M. Moore, "Ontological Excavation: Unearthing the core concepts of the application," presented at Working Conference on Reverse Engineering, Victoria, Canada, 2003.
[100] P. Hsia, A. Davis, and D. Kung, "Status Report: Requirements Engineering," *IEEE Software*, vol. 11, pp. 12-16, 1993.
[101] X. Huang and W. Lai, "Identification of clusters in the Web graph based on link topology," presented at Seventh International Database Engineering and Applications Symposium, 2003.
[102] J. Hughes, J. O'Brien, T. Rodden, M. Rouncefield, and I. Sommerfield, "Presenting Ethnography in the Requirements Process," presented at 2nd IEEE International Symposium on Requirements Engineering, 1995.
[103] E. Hutchins, *Cognition in the Wild*. Cambridge, MA: MIT Press, 1995.
[104] S. Iacono and R. Kling, "Computerization, Office Routines, and Changes in Clerical Work," in *Computerization and Controversy: Value Conflicts and Social Choices*, R. Kling, Ed., 2nd ed. New York, NY: Academic Press, 1996, pp. 309-315.
[105] M. Jackson, *Problem Frames: Analyzing and Structuring Software Development Problems*. New York, NY: Addison-Wesley, 2001.
[106] M. Jackson, *Software Requirements and Specifications: A Lexicon of Practice, Principles, and Prejudices*. New York, NY: Addison-Wesley, 1995.
[107] J. Jacky, *The Way of Z: Practical Programming with Formal Methods*. New York, NY: Cambridge University Press, 1997.
[108] E. Jacob, "Qualitative Research Traditions: A Review," *Review of Educational Research*, vol. 57, pp. 1-50, 1987.
[109] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Reading, MA: Addison-Wesley, 1999.
[110] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data Clustering: A Review," *ACM Computing Surveys*, vol. 31, pp. 264-323, 1999.
[111] D. R. Jeffery, G. C. Low, and M. Barnes, "A Comparison of Function Point Counting Techniques," *IEEE Transactions on Software Engineering*, vol. 19, pp. 529-532, 1993.

[112] I. John, D. Muthig, P. Sody, and E. Tolzmann, "Efficient and Systematic Software Evolution Through Domain Analysis," presented at IEEE Joint International Conference on Requirements Engineering (RE'02), 2002.
[113] W. L. Johnson and M. Feather, "Building an Evolution Transformation Library," presented at 12th International Conference on Software Engineering, Nice, France, 1990.
[114] C. Jones, *Assessment and Control of Software Risks*. Upper Saddle River, NJ: Prentice-Hall, Inc., 1994.
[115] L. G. Jones and A. L. Soule, "Software Process Improvement and Product Line Practice: CMMI and the Framework for Software Product Line Practice," Carnegie Mellon, Pittsburgh, PA CMU/SEI-2002-TN-012, 2002.
[116] H. Kaindl, "An Integration of Scenarios with their Purposes in Task Modeling," presented at Designing Interactive Systems: Processes, Practices, Methods, and Techniques, Ann Arbor, MI, 1995.
[117] K. C. Kang, "Feature-Oriented Development of Applications for a Domain," presented at Fifth International Conference on Software Reuse, 1998.
[118] K. C. Kang, J. Lee, and P. Donohoe, "Feature-Oriented Product Line Engineering," *IEEE Software*, vol. 19, pp. 58-65, 2002.
[119] M. Keil, P. Beranek, and B. Konsynski, "Usefulness and ease of use: field study evidence regarding task considerations," *Decision Support Systems*, vol. 13, pp. 75-91, 1995.
[120] C. F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Communications of the ACM*, vol. 30, pp. 416-429, 1987.
[121] C. F. Kemerer and S. Slaughter, "An Empirical Approach to Studying Software Evolution," *IEEE Transactions on Software Engineering*, vol. 25, pp. 493-509, 1999.
[122] A. Kirlik, *Requirements for Psychological Models to Support Design: Towards Ecological Task Analysis*, vol. 1. Hillsdale, NJ: Lawrence Erlbaum, 1995.
[123] H. Kohl, *From Archetuype to Zetigeist*. Boston, MA: Little, Brown and Company, 1992.
[124] M. Kolberg, E. Magill, D. Marples, and S. Tsang, "Feature Interactions in Services for Internet Personal Appliances," presented at IEEE International Conference on Communications, 2002.
[125] J. Kuusela and J. Savolainen, "Requirements Engineering for Product Families," presented at 22nd International Conference on Software Engineering, Limerick, Ireland, 2000.
[126] B. Laurel, "The Art of Human-Computer Interface Design," B. Laurel, Ed. Reading, MA: Addison-Wesley, 1990.
[127] A. L. Lederer, "The Role of Ease of Use, Usefulness, and Attitude in the Prediction of World Wide Web Usage," presented at ACM SIGCPR conference on Computer Personnel REsearch, Boston, MA, 1998.
[128] M. Lehman, "Laws of software evolution revisited," presented at 5th European Workshop on Software Process Technology, Nancy, France, 1996.
[129] M. Lehman and L. Belady, *Program Evolution: Processes of Software Change*, 1st ed. Orlando: Academic Press,inc., 1985.
[130] M. M. Lehman, "Software's Future: Managing Evolution," *IEEE Software*, vol. 15, pp. 40-44, 1998.
[131] M. M. Lehman, D. E. Perry, and J. F. Ramil, "Implications of evolution metrics on software maintenance," presented at International Conference on Software Maintenance, Bethesda, MD, 1998.
[132] M. M. Lehman and J. F. Ramil, "The Impact of Feedback in the Global Software Process," presented at Workshop on Software Process Simulation and Modeling (ProSim '98), Silver Falls, OR, 1998.
[133] M. M. Lehman, J. F. Ramil, P. D. Wemick, D. E. Perry, and W. M. Turski, "Metrics and Laws of Software Evolution - The Nineties View," presented at Fourth International Software Metrics Symposium, Albuquerque, NM, 1997.
[134] T. R. Leishman and D. A. Cook, "Requirements Risks Can Drown Software Projects," *Crosstalk*, vol. 15, pp. 4-8, 2002.
[135] D. B. Lenat, "CYC: A Large-Scale Investment in Knowledge Infrastructure," *Communications of the ACM*, vol. 38, pp. 33-48, 1995.
[136] N. G. Leveson, "Intent Specifications: An Approach to Building Human-Centered Specifications," *IEEE Transcations on Software Engineering*, vol. 26, pp. 15-35, 2000.
[137] Y. S. Lincoln and E. G. Guba, *Naturalistic Inquiry*. London, UK: Sage Publications, 1985.
[138] Luqi, "A Graph Model for Software Evolution," *IEEE Transactions On Software Engineering*, vol. 16, pp. 917-927, 1990.
[139] A. Maccari, "Experiences in Assessing Product Family Software Architecture for Evolution," presented at 24th International Conference on Software Engineering, 2002.

[140] A. Mackenzie, A. S. Ball, and S. R. Virdee, *Instant Notes in Ecology*. Oxford: BIOS Scientific Publishers Ltd, 1998.
[141] S. Maguire, *Writing Solid Code*. Redmond, WA: Microsoft Press, 1993.
[142] S. McConnell, *Code Complete*. Redmond, WA: Microsoft Press, 1993.
[143] S. McConnell, *Rapid Development*. Redmond, WA: Microsoft Press, 1996.
[144] J. McGrenere, ""Bloat": The Objective and Subjective Dimensions," presented at Computer Human Interaction 2000 (CHI 2000), 2000.
[145] J. McGrenere, R. M. Baecker, and K. S. Booth, "An Evaluation of a Multiple Interface Design Solution for Bloated Software," presented at CHI 2002, Minneapolis, MN, 2001.
[146] J. McGrenere and G. Moore, "Are We All In The Same "Bloat"?," presented at Graphics Interface 2000, Montreal, 2000.
[147] D. L. McGuinness, "Conceptual Modeling for Distributed Ontology Environments," presented at The Eight International Conference on Conceptual Structures Logical, Linguistic, and Computational Issues, Darmstadt, Germany, 2000.
[148] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder, "The Chimaera Ontology Environment," presented at Seventeenth National Conference on Artificial Intelligence, Austin, Texas, 2000.
[149] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder, "An Environment for Merging and Testing Large Ontologies," presented at Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000), Breckenridge, Colorado, 2000.
[150] A. Memon, I. Banerjee, and A. Nagarajan, "GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing," presented at Tenth Working Conference on Reverse Engineering, Victoria, BC Canada, 2003.
[151] T. Mens and S. Demeyer, "Future Trends in Software Evolution Metrics," presented at 4th International Workshop on Principles of Software Evolution, Vienna, Austria, 2002.
[152] M. Moore, "A Survey of Representations for Recovering User Interface Specifications in Reengineering," College of Computing, Georgia Institute of Technology, Atlanta, GA July 16, 1996 1996.
[153] M. Moore, "User Interface Reengineering," in *College of Computing*. Atlanta, GA: Georgia Institute of Technology., 1998.
[154] B. A. Myers, "User Interface Software Tools," *ACM Transactions on Computer-Human Interaction*, vol. 2, pp. 64-103, 1995.
[155] B. A. Nardi, "Studying Context: A Comparison of Activity Theory, Situated Action Models, and Distributed Cognition," in *Context and Consciousness: Activity Theory and Human-Computer Interaction*, B. A. Nardi, Ed. Cambridge, MA: MIT Press, 1996, pp. 69-102.
[156] B. A. Nardi and V. L. O'Day, *Information Ecologies: Using Technology with Heart*, Reprint Edition ed. Cambridge, MA: MIT Press, 2000.
[157] J. Nielsen, *Usability Engineering*. Cambridge, MA: Academic Press, 1993.
[158] J. Nielsen, "The Usability Engineering Life Cycle," *IEEE Computer*, vol. 25, pp. 12-22, 1992.
[159] G. M. Nijssen and T. A. Halpin, *Conceptual Schema and Relational Database Design*. New York: Prentice Hall, 1989.
[160] N. J. Nilsson, *Principles of Artifical Intelligence*. Palo Alto, CA: Tioga Publishing Company, 1980.
[161] D. Norman, *The Invisible Computer*. Cambridge, MA: MIT Press, 1998.
[162] D. A. Norman, *The Design of Everyday Things*. New York, NY: Doubleday, 1988.
[163] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap," presented at Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, 2000.
[164] P. Oreizy, "A Flexible Approach to Decentralized Software Evolution," presented at 1999 International Conference on Software Engineering, Los Angeles, CA, 1999.
[165] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Worl, "An Architecture-based Approach to Self-adaptive Software," *IEEE Intelligent Systems*, vol. 14, pp. 54-62, 1999.
[166] D. E. Perry, "Dimensions of Software Evolution," presented at International Conference on Software Maintenance, Victoria, BC Canada, 1994.
[167] H. Petroski, *The Book on the Book*. New York: Alfred A. Knopf, 1999.
[168] H. Petroski, *Design Paradigms: Case Histories of Error and Judgment in Engineering*. Cambridge: Cambridge University Press, 1994.
[169] H. Petroski, *The Evolution of Useful Things*, 1 ed. New York: Vintage Books, 1992.

[170] H. Petroski, *Invention by Design: How Engineers Get From Thought to Thing*. Cambridge, MA: Harvard University Press, 1996.
[171] H. Petroski, *The Pencil: A History of Design and Circumstance*. New York, NY: Alfred A. Knopf, 1992.
[172] C. Potts, "Requirements Models in Context," presented at 3rd International Symposium on Requirements Engineering (RE'97), Annapolis, MD, 1997.
[173] C. Potts, "Software Engineering Research Revisited," *IEEE Software*, vol. 10, pp. 19-26, 1993.
[174] C. Potts, "Using Schematic Scenarios to Understand User Needs," presented at Designing Interactive Systems: Processes, Practices, Methods, and Techniques, Ann Arbor, MI, 1995.
[175] C. Potts, A. Anton, and K. Takahashi, "Inquiry-Based Requirements Analysis," in *IEEE Software*, vol. 2, 1994, pp. 21-32.
[176] C. Potts and I. Hsi, "Abstraction and context in requirements engineering: Toward a Synthesis," *Annals of Software Engineering*, vol. 3, pp. 23-61, 1997.
[177] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 4th ed. New York, NY: McGraw Hill, 1997.
[178] M. E. Raichle, "Visualizing the Mind," *Scientific American*, vol. 270, pp. 58-64, 1994.
[179] E. S. Raymond, *The Cathedral and the Bazaar*. Sebastopol, CA: O'Reilly and Associates, 1999.
[180] E. M. Rogers, *Diffusion of Innovation*, 4th ed. New York: The Free Press, 1995.
[181] S. Rugaber, "Program Comprehension," *Encyclopedia of Computer Science and Technology*, vol. 35, pp. 341-368, 1995.
[182] S. Rugaber and M. Guzdial, "Ectropic Software," presented at Workshop on Software Change and Evolution, Los Angeles, CA, 1999.
[183] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
[184] R. Sedgewick, *Algorithms in C: Part 5*, 3rd ed. New York, NY: Addison-Wesley, 2002.
[185] E. E. Smith and D. L. Medin, *Categories and Concepts*. Cambridge, MA: Harvard University Press, 1981.
[186] I. Sommerville, T. Rodden, P. Sawyer, R. Bentley, and M. Twidale, "Integrating Ethnography Into The Requirements Engineering Process," presented at IEEE International Symposium on Requirements Engineering, San Diego, CA, 1992.
[187] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*. New York, NY: John Wiley and Sons, 1997.
[188] J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*. Reading, MA: Addison-Wesley, 1984.
[189] J. P. Spradley, *The Ethnographic Interview*. New York, NY: Harcourt Brace Jovanovich College Publishers, 1979.
[190] E. Stroulia, M. El-Ramly, and P. Sorenson, "From Legacy to Web through Interaction Modeling," presented at International Conference on Software Maintenance, Montréal, Canada, 2002.
[191] E. Stroulia and R. V. Kapoor, "Reverse Engineering Interaction Plans for Legacy Interface Migration," presented at 4th International Conference on Computer Aided Design of User Interfaces, Valenciennes, France, 2002.
[192] Sun Microsystems,"A Visual Index to the Swing Components,"From, available at http://java.sun.com/docs/books/tutorial/uiswing/components/components.html, 2004.
[193] A. Sutcliffe, Maiden, Minocha, and Manuel, "Supporting Scenario-based Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 24, pp. 1072-1088, 1998.
[194] S. M. Sutton, Jr. and L. J. Osterweil, "Product Families and Process Families," presented at 10th International Software Process Workshop, Dijon, France, 1996.
[195] C. R. Symons, "Function Point Analysis: Difficulties and Improvements," *IEEE Transactions on Software Engineering*, vol. 14, pp. 2-11, 1988.
[196] C. R. Turner, A. Fuggetta, L. Lavazza, and A. L. Wolf, "A Conceptual Basis for Feature Engineering," *Journal of Systems and Software*, vol. 49, pp. 3-15, 1999.
[197] C. R. Turner, A. Fuggetta, L. Lavazza, and A. L. Wolf, "Feature Engineering," presented at Ninth International Workshop on Software Specification and Design, 1998.
[198] A. van Duresen and T. Kuipers, "Identifying Objects Using Cluster and Concept Analysis," presented at International Conference on Software Engineering, Los Angeles, 1999.
[199] D. K. Van Duyne, J. A. Landay, and J. Hong, *The Design of Sites*. New York, NY: Addison-Wesley, 2003.

[200] G. M. A. Verheijen and J. Van Bekkum, "NIAM: An Information Analysis Method," in *Information Systems Design Methodologies*, T. W. Olle, H. G. Sol, and A. A. Verrijn-Stuart, Eds. Amsterdam: North-Holland Publishing Company, 1982.
[201] K. J. Vicente, "A Few Implications of an Ecological Approach to Human Factors," *Human Factors Society Bulletin*, vol. 33, pp. 1-4, 1990.
[202] K. J. Vicente and J. Rasmussen, "The Ecology of Human-Machine Systems II: Mediating "Direct Perception" in Complex Work Domains," *Ecological Psychology*, vol. 2, pp. 207-249, 19990.
[203] Y. Wand, V. C. Storey, and R. Weber, "An Ontological Analysis of the Relationship Construct in Conceptual Modeling," *ACM Transactions on Database Systems*, vol. 24, pp. 494-528, 1999.
[204] Y. Wand and R. Y. Wang, "Anchoring Data Quality Dimensions in Ontological Foundations," *Communications of the ACM*, vol. 39, pp. 86-95, 1996.
[205] S. Wasserman and K. Faust, *Social Network Analysis*. Cambridge: Cambridge University Press, 1994.
[206] J. M. Wing, "A Specifier's Introduction to Formal Methods," *IEEE Computer*, vol. 23, pp. 8,10-22,24, !990.
[207] N. Wirth, "A Plea for Lean Software," *IEEE Computer*, vol. 28, pp. 64-68, 1995.
[208] L. Wittgenstein, *Philosophical Investigations*. Oxford: Basil Blackwell, 1953.
[209] K. Yue, "Validating System Requirements by Functional Decomposition and Dynamic Analysis," presented at 11th International Conference on Software Engineering, Pittsburgh, PA, 1989.
[210] P. Zave, "Feature Interactions and Formal Specifications in Telecommunications," *Computer*, vol. 26, pp. 20-28, 30, 1993.

Appendix 1 – Introduction to the Case Studies

The following Appendices contain the case studies applying ontological excavation and use case silhouetting to the following applications:

- Appendix 2 – Windows 95/98 CD Player
- Appendix 3 – Palm Pilot 2000 Scheduler
- Appendix 4 – Protocol Calendar / Calculator
- Appendix 5 – Microsoft Notepad

Appendix 6 contains a follow-up case study to Microsoft Notepad where we made substantial changes to the ontology that we recovered.

The case studies have the following subsections:

- *Introduction* – Describes the application and its general purpose.
- *Modeling Issues* – During the course of ontological excavation, we encounter potential modeling issues which may affect our analysis. We describe them here.
- *Ontological Analysis* – This summarizes our findings after analyzing the ontology – a list of core concepts* ordered by centrality value, teleons that we identified, and a list of metrics that we measured from the ontology.
- *The Use Case Silhouette* – This summarizes our findings from developing a use case silhouette on the application from a set of use case obtained from that application's help files.
- *Morphology* – This shows a figure of the application's morphological map (or a portion of the map) and a list of elements in the morphological map. The diagrams are often very large (spanning several pages) and are not intended to be readable in this document).
- *Ontology* – This shows a figure of the application's ontology and a list of concepts in the ontology.
- *Conclusion* – We highlight the relevant findings from the case study and draw some conclusions from our results.

* Some of the concepts are in brackets ('[ ]'s). We use brackets in the cases where the name is unavailable from the morphology or correct modeling requires us to name a concept.

Appendix 2 – The Windows 95/98 CD Player Case Study

2.1 Introduction

This appendix describes the results of the ontological excavation, ontological analysis, and use case silhouette analysis of the Windows 95/98 CD Player. The Windows CD Player (Figure 9) allows the user to play CDs, to manage information about that CD, which has to be entered manually by the user, and to manage custom playlists.

Figure 9 – The Win 95/98 CD Player

2.2 Ontological Analysis

Below is a summary of the findings from the ontological analysis. They include the following information:

- List of core concepts and their centrality values.
- Subgroups identified by k-core analysis
- Statistics of the ontology

2.2.1 Core Concepts Identified

Core concepts are those concepts essential to that application's ontology. Table 11 shows a list of the concepts identified in the ontology of the application. Values have a range from 0 to 100 where 100 means that the concept has connections to all other concepts in the ontology and 0 means the concept is either an isolate or a leaf node in the ontology. Core concepts have a centrality value greater than or equal to 7.0 and have been italicized.

Table 11 – CD Player Concepts ordered by Centrality Value

Concept Name	Centrality	Description
[Current Track]	48.7	The track being played
[Play Mode]	44.7	The settings applied to how the current disc is to be played
Track	36.6	A playable unit on a compact disc
Disc	28.7	The compact disc
[Current Disc]	23.4	The compact disc currently being played
Playlist	19.5	The list of tracks in the order that they should be played
Artist	0	The name of the artist associated with the CD
Title	0	The title of the CD
Drive	0	The drive where the current CD resides
Random Order	0	A play setting that causes tracks of a playlist to be played randomly
Continuous Order	0	A play setting that causes tracks of a playlist to be played continuously
Intro Mode	0	A setting that plays a few seconds of a track

Concept Name	Centrality	Description
Track Name	0	The name of the track
Track Number	0	The number of the track on the CD
[Playing_or_Paused]	0	Whether the track is currently playing or is paused
Intro Play Length	0	The number of seconds for intro mode
Track Time Elapsed	0	The amount of time that a track has been playing
Track Time Remaining	0	The amount of time left on a playing track
Track Time	0	The total duration of a track
Total Playtime	0	The total playtime of a playlist
Playlist Time Remaining	0	The amount of time left on a playlist.

2.2.2 Subgroups Identified

Subgroups of concepts may suggest teleons. We used a *k-core analysis* to identify potential teleons in the ontology. A *k-core* is a connected, maximal, induced subgraph of nodes such that each node has a minimum degree greater than equal to *k*. Subgroups identified in the application are listed by their k-value in Table 12 along with the concepts contained in that subgroup.

Table 12 – CD Player Subgroups Identified by K-Core Analysis

k-value	Concepts in Subgroup
2	[Current Track], [Play Mode], Track, Disc, [Current Disc], Playlist

2.2.3 Statistics

The following table (Table 13) lists the overall composition of the ontology.

Table 13 – CD Player Ontological Metrics

# of entity types:	6
# of attributes:	14
# of nodes in ontology:	20
# of core concepts in ontology:	6
% of total ontology covered by core concepts:	30%
% of total ontology covered by peripheral concepts:	70%
% of ontology (no attributes) covered by core concepts:	100%
% of ontology (no attributes) covered by peripheral concepts:	0
Average centrality of concepts	9.60
Density (number of edges divided by total number of possible edges)	.11

2.3 The Use Case Silhouette

The use case silhouette process takes a set of use cases and uses them to obtain statistics such as the number of concepts present in the ontology and the amount of ontological coverage by those concepts. These findings are summarized in Table 14.

Table 14 – CD Player Use Case Silhouette Statistics

Source	Help file associated with application
# of use cases:	23
# concepts invoked:	16
ontological coverage:	80%

2.3.1 Ontological Coverage by Use Case

Table 15 lists the number of concepts, the number of unique concepts activated in each use case, and the coverage of the unique concepts with respect to the overall ontology. It also measures the proportion of core concepts found in that use case (including repeated references).

Table 15 – CD Player – Use Case Overview

#	Name	# of concepts	# of unique concepts	% of ontology	% core concepts
1	adding Tracks to Play lists	6	5	24%	50%
2	back button	1	1	5%	100%
3	CD-Rom Drives, using multiple	2	2	10%	0%
4	CDs: changing tricks	2	1	5%	100%
5	CDs: pausing	4	2	10%	50%
6	CDs: play lists	6	6	29%	44%
7	CDs: storing track titles	6	6	29%	38%
8	changing: settings	6	6	29%	17%
9	clearing play lists	3	3	14%	67%
10	deleting tracks from play lists	4	3	14%	50%
11	ejecting CDs	1	1	5%	100%
12	forward button	1	1	5%	100%
13	moving between tracks	2	2	10%	100%
14	multidisc play	1	1	5%	0%
15	next track	1	1	5%	100%
16	options	6	6	29%	60%
17	previous track	1	1	5%	100%
18	random order	1	2	10%	50%
19	resetting play lists	5	2	10%	100%
20	resuming play	2	2	10%	50%
21	rewinding	1	1	5%	100%
22	skipping tracks	2	1	5%	100%
23	stop a CD	1	1	5%	100%

2.3.2 Concept Frequency Across Use Cases

Concept frequency looks at how often a concept is accessed across all the use cases and how often it is accessed against all the concepts invoked by all of the use cases. These are summarized in Table 16. Concept frequency is used to compare against a concept's centrality measures to see whether it retains its importance in the set of use cases. Presumably, a discrepancy would indicate that a concept with structural importance but lacking importance relative to actual usage needs to be made more prominent in the morphology, less prominent in the ontology, or is a symptom of a discontinuity between the system's model of usage and the goals of its users.

Table 16 – CD Player Frequency of Concept appearance in use case set. Core concepts are italicized.

Name	# Times Accessed	% of Total # of concepts invoked
Playlist	18	26 %
[Current Track]	10	14 %
[Current Disc]	8	11 %
Track	8	11 %
[Play Mode]	7	6 %
Artist	4	6 %
Title	3	4 %
Track Name	3	4 %
[Playing_or_Paused]	3	4 %
Track Number	2	3 %
Disc	1	1 %

Name	# Times Accessed	% of Total # of concepts invoked
Drive	1	1 %
Random Order	1	1 %
Intro Play Length	1	1 %
Track Time Elapsed	1	1 %
Track Time Remaining	1	1 %
Track Time	1	1 %
Continuous Order	0	0 %
Intro Mode	0	0 %
Total Playtime	0	0 %
Playlist Time Remaining	0	0 %

2.4 Morphology

Figure 10 – CD Player Morphological Map

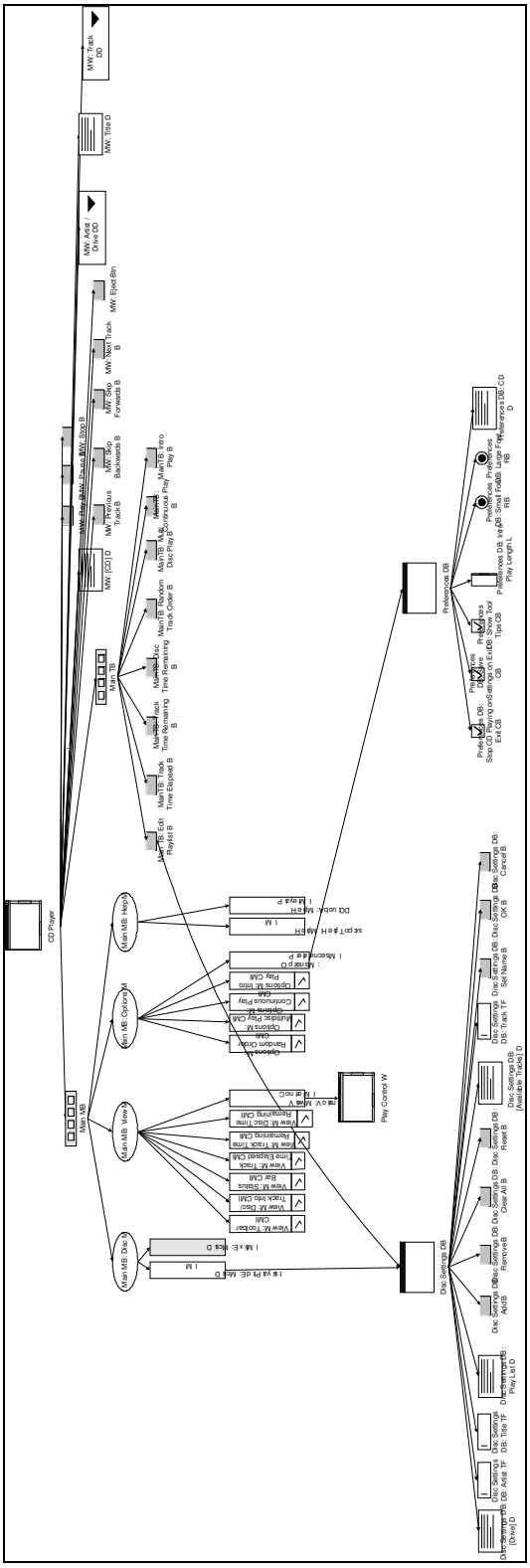


Figure 10 shows the morphological map of the application. Table 17 contains a list of the following morphological elements. They are numbered by the order that they were placed into the diagram.

Table 17 – CD Player Morphological Elements

#	Name of Morphological Element
1	CD Player
2	Main MB: Disc M
3	Main MB: View M
4	Main MB: Options M
5	Main MB: Help M
6	Disc M: Edit Playlist MI
7	Disc M: Exit MI
8	View M: Toolbar CMI
9	View M: Disc/Track Info CMI
10	View M: Status Bar CMI
11	View M: Track Time Elapsed CMI
12	View M: Track Time Remaining CMI
13	Options M: Random Order CMI
14	Options M: Multidisc Play CMI
15	Options M: Continuous Play CMI
16	Options M: Intro Play CMI
17	Options M: Preferences MI
18	Help M: Help Topics MI
19	Help M: About CD Player MI
20	View M: Volume Control MI
21	Play Control W
22	Main TB
23	MW: Artist / Drive DD
24	MW: Track DD
25	MW: [CD] D
26	MW: Play B
27	MW: Pause B
28	MW: Stop B
29	MW: Previous Track B
30	MW: Skip Backwards B
31	MW: Skip Forwards B
32	MW: Next Track B
33	MW: Eject Btn
34	Main TB: Edit Playlist B

#	Name of Morphological Element
35	MainTB: Track Time Elapsed B
36	MainTB: Track Time Remaining B
37	MainTB: Disc Time Remaining B
38	MainTB: Random Track Order B
39	MainTB: Multi Disc Play B
40	MainTB: Continuous Play B
41	MainTB: Intro Play B
42	MW: Title D
43	Disc Settings DB
44	Disc Settings DB: [Drive] D
45	Disc Settings DB: Artist TF
46	Disc Settings DB: Title TF
47	Disc Settings DB: Play List D
48	Disc Settings DB: Add B
49	Disc Settings DB: Remove B
50	Disc Settings DB: Clear All B
51	Disc Settings DB: Reset B
52	Disc Settings DB: [Available Tracks] D
53	Disc Settings DB: Track TF
54	Disc Settings DB: Set Name B
55	Disc Settings DB: OK B
56	Disc Settings DB: Cancel B
57	Preferences DB
58	Preferences DB: Stop CD Playing on Exit CB
59	Preferences DB: Save Settings on Exit CB
60	Preferences DB: Show Tool Tips CB
61	Preferences DB: Intro Play Length L
62	Preferences DB: CD D
63	Main MB
64	View M: Disc Time Remaining CMI
65	Preferences DB: Small Font RB
66	Preferences DB: Large Font RB

2.5 Ontology

Figure 11 – CD Player Ontology

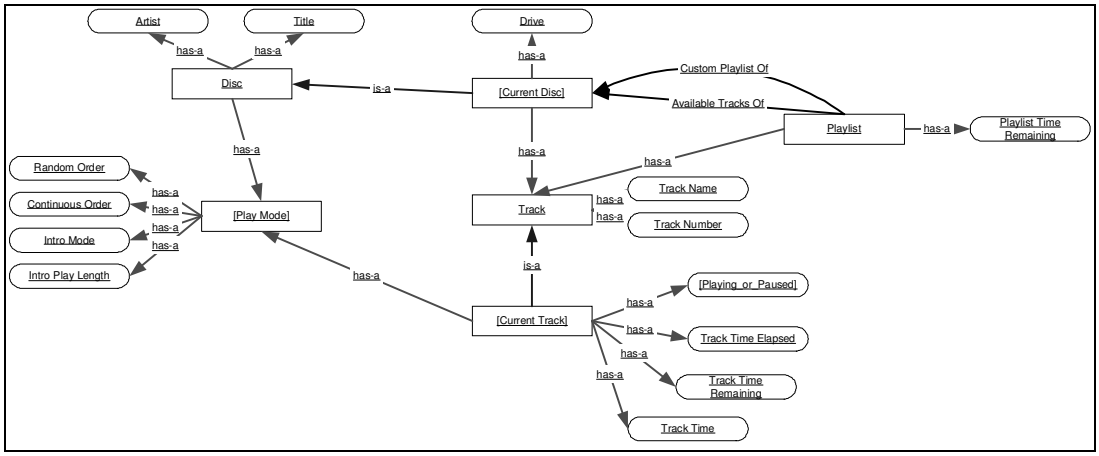


Figure 11 shows the ontology for the application. Table 18 shows a list of the concepts identified in the ontology. They are numbered by the order that they were placed into the diagram and identified as an Entity type or an Attribute (attributes are concepts that lack independent identity from the connected entity type)

2.5.1 Concepts in Application

Table 18 – CD Player Ontological Elements. The letter E indicates the concept is an Entity type. The letter A indicates the concept is an Attribute.

#	Concepts	Type
1	Disc	E
2	Artist	A
3	Title	A
4	Current Disc	E
5	Drive	A
6	Play Mode	E
7	Random Order	A
8	Continuous Order	A
9	Intro Mode	A
10	Track	E
11	Track Name	A

#	Concepts	Type
12	Track Number	A
13	Current Track	E
14	Playing_or_Paused	A
15	Playlist	E
16	Intro Play Length	A
17	Track Time Elapsed	A
18	Track Time Remaining	A
19	Track Time	A
20	Total Playtime	A
21	Playlist Time Remaining	A

2.6 Conclusion

The CD Player has a very simple ontology where all the entity types are also core concepts. As a result, it demonstrates a very high conceptual coherence across all the metrics and seems to be a good candidate for a Reef ontological structure. The centrality metrics show Current Track to be most important concept in the graph, which we expect given that a user playing a CD is likely to be most interested in the track currently playing. However, the use case silhouettes show Playlist to be the most important concept. We attribute this to the number of use cases in the Help files that involve Playlist. Managing the playlist of for the CD requires a lot of steps. However, the playlist use cases only invoke 44% of the core concepts which show that it this use case is less likely to be used than any of the other use cases that are composed entirely of core concepts.

Appendix 3 – Palm Pilot Scheduler Case Study

3.1 Introduction

This appendix describes the results of the ontological excavation, ontological analysis, and use case silhouette analysis of the Palm Pilot Scheduler. The Palm Pilot Scheduler lived on the Palm Pilot 2000 and provided its user with features such as event scheduling, alarms, and synchronization with other applications. The ontology was recovered by Colin Potts. Missing from this analysis are the morphological map and use case silhouettes as those methods were developed after the recovery of this ontology.

3.2 Ontological Analysis

Below is a summary of the findings from the ontological analysis. They include the following information:

- List of core concepts and their centrality values.
- Subgroups identified by k-core analysis
- Statistics of the ontology

3.2.1 Core Concepts Identified

Core concepts are those concepts essential to that application's ontology. Table 19 shows a list of the concepts identified in the ontology of the application. Values have a range from 0 to 100 where 100 means that the concept has connections to all other concepts in the ontology and 0 means the concept is either an isolate or a leaf node in the ontology. Core concepts have a centrality value greater than or equal to 7.0 and have been italicized.

Table 19 – Palm Pilot Scheduler Concepts ordered by Centrality Value

Concept Name	Value	Description
Event	46.2	A schedulable item
Date	29.7	The day, month, and year of the event
To Do Item	29.7	A task that needs to be completed
Hot Synch	18.9	Dynamic synchronization with another computer
Day	16.4	A day of the month
Month	14.2	A month of the year
Time	11.6	Hour, Minute, Second
Alarm	10.6	A timed alert for events
Repetition	9.5	A setting
Note	7.0	Text describing the event or to-do list
Every	6.5	A repetition setting for events that take place on the same day of the week
Start Time	5.0	The start time of the item
End Time	5.0	The end time of the item
Application	4.8	The application synching with Scheduler
Alarm Units	3.9	The time settings for the alarm
PurgeUnits	3.8	The time settings for the monthly memory purges
Priority	3.5	The importance of the item
Synch Status	3.5	The progress of the synchronization
Due Date	3.4	The date a to-do item must be completed
To Do List	2.8	The list of tasks that need to be completed
Synch Problem	2.0	A notification that an error has occurred during synchronization

Concept Name	Value	Description
To Do Problem	1.2	An error notification with synching a to-do problem
Event Problem	1.1	An error notification with synching an event problem
Today	0.8	The current day
Week	0.6	The current week of the month
Preferences	0.6	Preference settings for the scheduler
Schedule	0.5	The settings for a repetition.
End Date	0.4	The day that an event stops repeating
Hour	0.4	The hour of the day
Minute	0.4	The minute of the hour
Year	0.3	The year of interest
Backup Copy	0.3	Backup copies for Event and To-Do Item
Purge	0	The scheduled event that clears finished events and tasks from memory
Month Number	0	The number of a month of the year
Month Name	0	The name of a month of the year
Frequency	0	The interval with which an event repeats
Day Number	0	The day of a month by number
Day Name	0	The name of a day by week
All Occurrences	0	A setting that determines whether an event repeats for all instances of that event
Current Occurrences	0	A setting that determines whether a change affects the current occurrence of the event.
OneThruFive	0	The range of a priority setting.
ToDoItemName	0	The name of a to-do item.
Event Name	0	The name of an event
Is Due	0	An attribute of a to-do item.
Is Private	0	Determines whether an event or to-do item is visible to other people
Note Topic	0	The heading that describes a note.
Is Scheduled	0	Whether an event is scheduled or not.
Is Preset	0	Whether an alarm is preset to announce any event or task
Latency	0	How long an alarm sounds before being shut off
Is Unfiled	0	Whether a task has been filed or not
Category	0	What group a task belongs to.
Is Done	0	Whether a task is finished or not.
Content	0	The information contained in a note.
AM PM	0	Whether an hour is a AM or PM.
Five Minutes	0	An interval for setting times
Date Book	0	The item in an application that stores synch data from the Scheduler
Last Hot Synch	0	The time the scheduler last synchronized its data with an external application
Is OK	0	The status of a synchronization.

3.2.2 Teleons Identified

Teleons suggest morphological features. We used a *k-core analysis* to identify potential teleons in the ontology. A *k-core* is a connected, maximal, induced subgraph of nodes such that each node has a minimum degree greater than equal to *k*. Teleons identified in the application are listed by their k-value in Table 20 along with the concepts contained in that subgraph.

Table 20 – Palm Pilot Scheduler Teleons Identified by K-Core Analysis

k-value	Concepts in Teleon
2	PurgeUnits, Week, Month, Every, Today, Day, Preferences, End Date, Repetition, Schedule, Year, Date, Due Date, All Occurrences, Current Occurrences, Event, To Do Item, Is Private, Start Time, End Time, Alarm, Alarm Units, Hour, Minute, Backup Copy, Note, Event Problem, Synch Problem, To Do Problem, To Do List, Hot Synch, Time, Application

3.2.3 Statistics

The following table (Table 21) lists the overall composition of the ontology.

Table 21 – Palm Pilot Scheduler Ontological Metrics

# of entity types:	43
# of attributes:	16
# of nodes in ontology:	58
# of core concepts in ontology:	10
% of total ontology covered by core concepts:	17 %
% of total ontology covered by peripheral concepts:	83 %
% of ontology (no attributes) covered by core concepts:	23 %
% of ontology (no attributes) covered by peripheral concepts:	77 %
Average centrality of concepts	4.22
Density (number of edges divided by total number of possible edges)	.05

3.3 The Use Case Silhouette

The use case silhouette process takes a set of use cases and uses them to obtain statistics such as the number of concepts present in the ontology and the amount of ontological coverage by those concepts. No use cases were available for this application.

3.4 Ontology

Figure 12 – Palm Pilot Scheduler Ontology

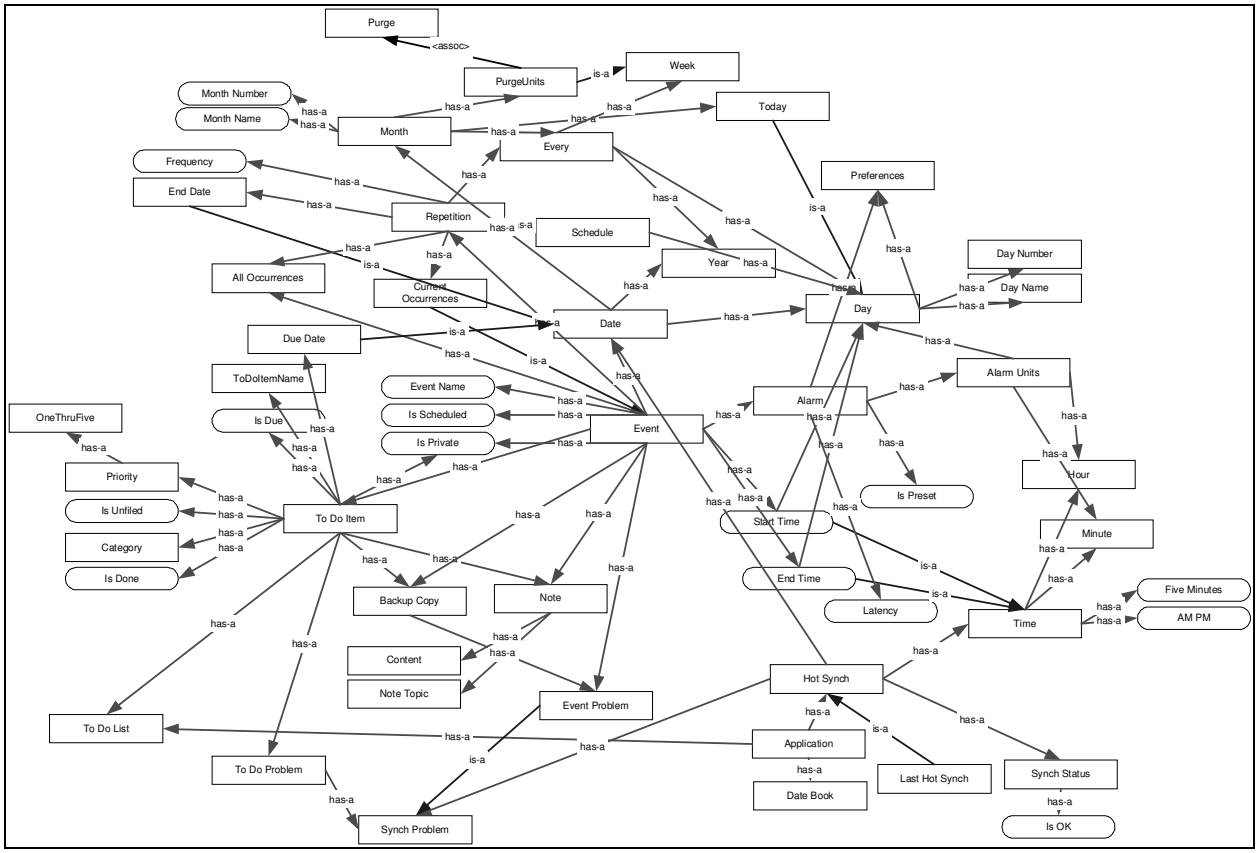


Figure 12 shows the ontology for the application. Table 22 shows a list of the concepts identified in the ontology. They are numbered by the order that they were placed into the diagram and identified as an Entity Type or an Attribute (attributes are concepts that lack independent identity from the connected entity type)

3.4.1 Concepts in Application

Table 22 – Palm Pilot Scheduler Ontological Elements. The letter E indicates the concept is an Entity type. The letter A indicates the concept is an Attribute.

#	Concept	Type
1	Purge	E
2	PurgeUnits	E
3	Week	E
4	Month Number	A
5	Month Name	A
6	Frequency	A
7	Month	E
8	Every	E
9	Today	E
10	Day	E

#	Concept	Type
11	Day Number	E
12	Day Name	E
13	Preferences	E
14	End Date	E
15	Repetition	E
16	Schedule	E
17	Year	E
18	Date	E
19	Due Date	E
20	All Occurrences	E

#	Concept	Type
21	Current Occurrences	E
22	OneThruFive	E
23	ToDoItemName	E
24	Event Name	A
25	Event	E
26	Priority	E
27	Is Due	A
28	To Do Item	E
29	Is Private	A
30	Note Topic	E
31	Start Time	A
32	End Time	A
33	Is Scheduled	A
34	Alarm	E
35	Alarm Units	E
36	Is Preset	A
37	Latency	A
38	Hour	E
39	Minute	E
40	Is Unfiled	A

#	Concept	Type
41	Category	E
42	Is Done	A
43	Backup Copy	E
44	Content	E
45	Note	E
46	Event Problem	E
47	Synch Problem	E
48	To Do Problem	E
49	To Do List	E
50	Hot Synch	E
51	Time	E
52	AM PM	A
53	Five Minutes	A
54	Application	E
55	Date Book	E
56	Last Hot Synch	E
57	Is OK	A
58	Synch Status	E

3.5 Conclusion

The Palm Pilot Scheduler was recovered using a slightly different modeling and recovery process than the other three applications covered in these case studies. What makes it interesting is the degree to which its concepts are interrelated such that the teleon analysis revealed one very large k-core. We expected that features such as synching the database with an external computer would have been more distinct in the ontology. The Scheduler also shows a lower conceptual coherence across all measures than CD Player, possibly due to the complexity of the scheduling model that it embodies. Based on the concepts it embodies, we believe Scheduler to have a Reef ontological structure but with the potential to become an Urban one given its complexity.

Appendix 4 – Protocol Calendar / Calculator Case Study

4.1 Introduction

This appendix describes the results of the ontological excavation, ontological analysis, and use case silhouette analysis of the Protocol Calendar / Calculator. The Protocol Calendar / Calculator (Figure 13) is a device that implements an alarm clock, calendar, calculator, currency exchange calculator, and countdown timer. The clock also allows its users to view times in sixteen different time zones.

Figure 13 – The Protocol Calendar / Calculator

4.2 Modeling Issues

The Protocol Calendar / Calculator has a Toolkit Ontological Structure. We performed the standard analysis plus a series of analyses for each subgroup. We asked ourselves whether the Time Zones should be modeled as entity types or instances (and they clearly could not be modeled as attributes of Time Zone as they do have independent existence). From a certain perspective, they are clearly instances as they have proper names and are not really generalizable (what is a type of "New York Time Zone"?). In the instance case, we simply ignore them in the ontology. An argument for modeling them as entity types is that they seem to be specifically chosen and, in the spirit of both black box and anthropological methods, we should consider them as important to the ontology of the device. For completeness, we present first the concepts modeled with Time Zones as first order objects and then the concepts and centrality values without Time Zones. We also modeled the calculator loosely. For example, a Calculator

implements Mathematical Operations such as Addition but nowhere in the ontology does it mention the ontology of the addition operation or that it is performed on numbers. We chose not to recover or re-derive the ontology of arithmetic and felt that our representations were sufficient. However, if one wanted to distinguish an ordinary financial calculator from a scientific one, it might be necessary to model the actual mathematical concepts that each embodies. Lastly, we modeled the Currency Exchange calculator as being independent of the actual calculator since it seemed to be a separate feature even though it is probably implemented through the embedded system of the calculator.

4.3 Ontological Analysis

Below is a summary of the findings from the ontological analysis. They include the following information:

- List of core concepts and their centrality values.
- Subgroups identified by k-core analysis
- Statistics of the ontology

4.3.1 Core Concepts Identified

Core concepts are those concepts essential to that application's ontology. Table 23 shows a list of the concepts identified in the ontology of the application. Values have a range from 0 to 100 where 100 means that the concept has connections to all other concepts in the ontology and 0 means the concept is either an isolate or a leaf node in the ontology. Core concepts have a centrality value greater than or equal to 7.0 and have been italicized.

Table 23 – Protocol Calendar / Calculator Concepts ordered by Centrality Value

Concept	Centrality	Description
[Time Zone]	30.3	One of the 24 longitudinal segments of the planet specifying an hour of time.
Time	21.3	hour : minute : second
Home Time	18.9	The time zone where the user of the device currently resides
[Time Display Mode]	5.1	The setting that determines whether the AM/PM or 24-hour time display is used
Alarm Time	4.9	The time that an alarm will sound
[Mathematical Operation]	4.2	An operation performed by the calculator
Alarm	3.2	The alarm concept including time of activation, sound, and whether it is on or not.
Count Down Timer	2.2	A timer that is set to a time quantity and sounds an alarm when that time has elapsed.
Hour	1.2	Sixty minutes
Minute	1.2	Sixty seconds
Second	1.2	Smallest unit of time.
Sound	0.5	A specific sequence of notes.
Date	0.3	The day and month.
Month	0.1	One of the 12 months in a year
Year	0.1	Unit of time specifying the interval for the planet to make one full rotation around the Sun
Calendar	0	The display that shows the current month and year.
New York [Time Zone]	0	The time zone of New York
12-hr Time Display	0	A setting that displays the time as having an AM and PM setting.
24-hr Time Display	0	A setting that displays the time as a 24 hour increment
[Addition Operation]	0	The '+' operation
[Division Operation]	0	The '÷' operation

Concept	Centrality	Description
[Equals Operation]	0	The '=' operation
[Memory Subtract Operation]	0	The 'M-' operation
[Memory Recall Operation]	0	The 'MRC' operation – recalls the currently
[Memory Save/Add Operation]	0	The 'M+' operation – saves a number or adds a number to the quantity previously saved
[Multiplication Operation]	0	The '×' operation
[Percent Operation]	0	The '%' operation
[Subtraction Operation]	0	The '-' operation
Alarm OnOrOff	0	A setting that determines whether an alarm is active or not.
Bangkok [Time Zone]	0	The time zone of Bangkok
Cairo [Time Zone]	0	The time zone of Cairo
Calculator	0	A device that performs mathematical operations on numbers
Chicago [Time Zone]	0	The time zone of Chicago
Currency Exchange [Calculator]	0	The calculation that converts one currency into another currency.
Denver [Time Zone]	0	The time zone of Denver
Exchange Rate	0	The numerical value used to calculate the value of one currency against another one.
Hong Kong [Time Zone]	0	The time zone of Hong Kong
Honolulu [Time Zone]	0	The time zone of Honolulu
Karachi [Time Zone]	0	The time zone of Karachi
London [Time Zone]	0	The time zone of London
Los Angeles [Time Zone]	0	The time zone of Los Angeles
Moscow [Time Zone]	0	The time zone of Moscow
Paris [Time Zone]	0	The time zone of Paris
Rio De Janeiro [Time Zone]	0	The time zone of Rio de Janeiro
Sydney [Time Zone]	0	The time zone of Sydney
Tokyo [Time Zone]	0	The time zone of Tokyo
Wellington [Time Zone]	0	The time zone of Wellington
Day	0	An increment of time that has a duration of 24 hours.

We performed a second analysis on the ontology to see what effect removing the concepts that described the sixteen time zones would have on the other concepts. Table 24 shows the results. Removing these concepts had the effect of lowering the overall centrality values for any of the concepts that concerned time.

Table 24 – Protocol Calendar / Calculator Concepts ordered by Centrality Value without Time Zones

Concept	Centrality
Time	11.7
[Mathematical Operation]	9.7
[Time Display Mode]	4.9
Alarm Time	3.5
Alarm	3.2
Home Time	2.6
Count Down Timer	2.6
Sound	1.1
Hour	0.8
Minute	0.8

Concept	Centrality
Second	0.8
Date	0.8
Month	0.2
Year	0.2
Calendar	0.1
12-hr Time Display	0
24-hr Time Display	0
[Addition Operation]	0
[Division Operation]	0
[Equals Operation]	0

Concept	Centrality
[Memory Subtract Operation]	0
[Memory Recall Operation]	0
[Memory Save/Add Operation]	0
[Multiplication Operation]	0
[Percent Operation]	0
[Subtraction Operation]	0

Concept	Centrality
Alarm OnOrOff	0
Calculator	0
Currency Exchange[Calculator]	0
Exchange Rate	0
[Time Zone]	0
Day	0

4.3.2 Concepts Organized By Subgroup (Toolkit Ontological Structure)

We identified this application as a toolkit – an application possessing several subgroups of concepts that are disconnected from one another. Below we list the subgroups and their concepts identified in the ontology. Specifically, we present the centrality values of the subgroup of concepts composing the Calendar (Table 25), Clock / Timer / Alarm (with Time Zones) (Table 26), Clock / Timer / Alarm (without Time Zones) (Table 27), Calculator (Table 28), and the Currency Exchange calculator (Table 29), respectively.

Table 25 – Protocol Calendar / Calculator – Calendar Subgroup Concepts ordered by Centrality Value

Concept	Centrality
Date	58.3
Month	16.7
Year	16.7
Calendar	8.3
Day	0

Table 26 – Protocol Calendar / Calculator – Time / Timer Subgroup Concepts ordered by Centrality Value

Concept	Centrality
[Time Zone]	80.8
Time	56.7
Home Time	50.2
[Time Display Mode]	13.5
Alarm Time	12.9
Alarm	8.5
Count Down Timer	5.9
Hour	3.2
Minute	3.2
Second	3.2
Sound	1.3
New York [Time Zone]	0
12-hr Time Display	0
24-hr Time Display	0
Alarm OnOrOff	0

Concept	Centrality
Bangkok [Time Zone]	0
Cairo [Time Zone]	0
Chicago [Time Zone]	0
Denver [Time Zone]	0
Hong Kong [Time Zone]	0
Honolulu [Time Zone]	0
Karachi [Time Zone]	0
London [Time Zone]	0
Los Angeles [Time Zone]	0
Moscow [Time Zone]	0
Paris [Time Zone]	0
Rio De Janeiro [Time Zone]	0
Sydney [Time Zone]	0
Tokyo [Time Zone]	0
30 Wellington [Time Zone]	0

Table 27 – Protocol Calendar / Calculator – Time / Timer Subgroup (no Time Zones) Concepts ordered by Centrality Value

Concept	Centrality
Time	69.6
[Time Display Mode]	29.5
Alarm Time	21.2
Alarm	18.9
Home Time	15.4
Count Down Timer	15.4
Sound	6.7
Hour	4.8
Minute	4.8
Second	4.8
12-hr Time Display	0
24-hr Time Display	0
Alarm OnOrOff	0
[Time Zone]	0

Table 28 – Protocol Calendar / Calculator Concepts – Calculator Subgroup – ordered by Centrality Value

Concept	Centrality
[Mathematical Operation]	100
[Addition Operation]	0
[Division Operation]	0
[Equals Operation]	0
[Memory Subtract Operation]	0
[Memory Recall Operation]	0
[Memory Save/Add Operation]	0
[Multiplication Operation]	0
[Percent Operation]	0
[Subtraction Operation]	0
Calculator	0

Table 29 – Protocol Calendar / Calculator Concepts – Currency Exchange Calculator Subgroup – ordered by Centrality Value

Concept	Centrality
Currency Exchange [Calculator]	0
Exchange Rate	0

4.3.3 Teleons Identified

Teleons suggest morphological features. We used a *k-core analysis* to identify potential teleons in the ontology. A *k-core* is a connected, maximal, induced subgraph of nodes such that each node has a minimum degree greater than equal to *k*. Teleons identified in the application are listed by their k-value in Table 30 along with the concepts contained in that subgraph.

Table 30 – Protocol Calendar / Calculator Teleons Identified by K-Core Analysis

k-value	Concepts in Teleon
2	Date, Month, Year, Calendar
2	Alarm, Alarm Time, Count Down Timer, Hour, Minute, Second, Sound, Time,

4.3.4 Statistics

Table 31 lists the overall composition of the ontology. Table 32 lists the composition of the ontology without the Time Zone entity types. We also include the statistics for the individual subgroups in Tables 44-48.

Table 31 – Protocol Calendar / Calculator Ontological Metrics

# of entity types:	47
# of attributes:	1
# of nodes in ontology:	48
# of core concepts in ontology:	3
% of total ontology covered by core concepts:	6 %
% of total ontology covered by peripheral concepts:	94 %
% of ontology (no attributes) covered by core concepts:	6 %
% of ontology (no attributes) covered by peripheral concepts:	94 %
Average centrality of concepts	1.97
Density (number of edges divided by total number of possible edges)	.04

Table 32 – Protocol Calendar / Calculator Ontological Metrics – No Time Zone

# of entity types:	32
# of attributes:	1
# of nodes in ontology:	33
# of core concepts in ontology:	2
% of total ontology covered by core concepts:	6 %
% of total ontology covered by peripheral concepts:	94 %
% of ontology (no attributes) covered by core concepts:	6 %
% of ontology (no attributes) covered by peripheral concepts:	94 %
Average centrality of concepts	1.34
Density (number of edges divided by total number of possible edges)	.06

Table 33 – Protocol Calendar / Calculator Ontological Metrics – Calendar Subgroup

# of entity types in subgroup:	5
# of nodes in subgroup ontology:	5
# of core concepts in subgroup ontology:	4
% of total ontology covered by subgroup:	10 %
% of total ontology (no time zones) covered by subgroup:	16 %
% of subgroup ontology covered by core concepts w/in subgroup:	80 %
% of subgroup ontology covered by peripheral concepts w/in subgroup:	20 %
Average centrality of concepts	20.0
Density (number of edges divided by total number of possible edges)	.50

Table 34 – Protocol Calendar / Calculator Ontological Metrics – Time Subgroup

# of entity types in subgroup:	29
# of nodes in subgroup ontology:	30
# of core concepts in subgroup ontology:	6
% of total ontology covered by subgroup:	63 %
% of subgroup ontology covered by core concepts w/in subgroup:	20 %
% of subgroup ontology covered by peripheral concepts w/in subgroup:	80 %
Average centrality of concepts	7.99
Density (number of edges divided by total number of possible edges)	.07

Table 35 – Protocol Calendar / Calculator Ontological Metrics – Time Subgroup – no Time Zones

# of entity types in subgroup:	13
# of nodes in subgroup ontology:	14
# of core concepts in subgroup ontology:	6
% of total ontology covered by subgroup:	44 %
% of subgroup ontology covered by core concepts w/in subgroup:	43 %
% of subgroup ontology covered by peripheral concepts w/in subgroup:	57 %
Average centrality of concept	13.64
Density (number of edges divided by total number of possible edges)	.18

Table 36 – Protocol Calendar / Calculator Ontological Metrics – Calculator Subgroup

# of entity types in subgroup:	11
# of nodes in subgroup ontology:	11
# of core concepts in subgroup ontology:	1
% of total ontology covered by subgroup:	21%
% of total ontology (no time zones) covered by subgroup:	31%
% of subgroup ontology covered by core concepts w/in subgroup:	10 %
% of subgroup ontology covered by peripheral concepts w/in subgroup:	90 %
Average centrality of concept	9.09
Density (number of edges divided by total number of possible edges)	.18

Table 37 – Protocol Calendar / Calculator Ontological Metrics – Currency Exchange Calculator Subgroup

# of entity types in subgroup:	2
# of nodes in subgroup ontology:	2
# of core concepts in subgroup ontology:	0
% of total ontology covered by subgroup:	4 %
% of total ontology (no time zones) covered by subgroup:	6 %
% of subgroup ontology covered by core concepts w/in subgroup:	0 %
% of subgroup ontology covered by peripheral concepts w/in subgroup:	100 %
Average centrality of concept	0
Density (number of edges divided by total number of possible edges)	1.00

4.4 The Use Case Silhouette

The use case silhouette process takes a set of use cases and uses them to obtain statistics such as the number of concepts present in the ontology and the amount of ontological coverage by those concepts. These findings are summarized in Table 38.

Table 38 – Protocol Calendar / Calculator Use Case Silhouette Statistics

Source	Instructions included with device
# of use cases:	11
# concepts invoked:	48
ontological coverage:	100%

4.4.1 Ontological Coverage by Use Case

Table 39 lists the number of concepts, the number of unique concepts activated in each use case, and the coverage of the unique concepts with respect to the overall ontology. It also measures the proportion of core concepts found in that use case (including repeated references). Because the application has also been identified as a Toolkit, the table also includes a breakdown of the use cases by subgroup including the ontological coverage of the use case concepts within the subgroup. In the case of the Time subgroup, we also included an analysis without the sixteen time zones.

Table 39 – Protocol Calendar / Calculator – Use Case Overview

#	Name	# of concepts	# of unique concepts	% of ontology	% ontology no Time Zone	% core concepts	Subgroup	% Subgroup Ontology	% Subgroup Ontology / No Time Zone
1	Calendar Setting	7	5	10 %	16 %	0 %	Calendar	100 %	N/A
2	Calendar / Time Setting Mode	30	13	27 %	41 %	15 %	Calendar / Time	37 %	68 %
3	Set HOME TIME	33	18	38 %	2 %	11 %	Time	60 %	95 %
4	Set Count-Down Timer	12	4	8 %	13 %	0 %	Time	13 %	21 %
5	Start / Stop / Zero Count-Down Timer	3	1	2 %	3 %	0 %	Time	3 %	5 %
6	Set Alarm	10	5	10 %	16 %	0 %	Time	17 %	26 %
7	Turn Alarm On / Off	1	1	2 %	3 %	0 %	Time	3 %	5 %
8	Set Alarm Sound	2	2	2 %	6 %	0 %	Time	7 %	11 %
9	Calculator	16	11	23 %	34 %	0 %	Calculator	100%	N/A
10	Currency Exchange Calculator	3	2	2 %	6 %	0 %	Currency Exchange	100%	N/A
11	Set Keytone On / Off	1	1	2 %	3 %	0 %	Time	3%	5 %

4.4.2 Concept Frequency Across Use Cases

Concept frequency looks at how often a concept is accessed across all the use cases and how often it is accessed against all the concepts invoked by all of the use cases. These are summarized in Table 40. Concept frequency is used to compare against a concept's centrality measures to see whether it retains its importance in the set of use cases. Presumably, a discrepancy would indicate that a concept with structural importance but lacking importance relative to actual usage needs to be made more prominent in the morphology, less prominent in the ontology, or is a symptom of a discontinuity between the system's model of usage and the goals of its users.

Table 40 – Protocol Calendar / Calculator Frequency of Concept appearance in use case set. Core concepts are italicized.

Name	# Times Accessed	% of Total # of concepts invoked
[Time Zone]	16	13%
Count Down Timer	9	8%
Hour	7	6%
Minute	7	6%
Second	7	6%
[Mathematical Operation]	6	5%
Month	6	5%
Time	5	4%
Year	5	4%
Alarm	3	3%
Date	3	3%
Calendar	3	3%
Day	3	3%
Home Time	2	2%
[Time Display Mode]	2	2%
Alarm Time	2	2%
Exchange Rate	2	2%
Sound	2	2%
New York [Time Zone]	1	1%
12-hr Time Display	1	1%
24-hr Time Display	1	1%
[Addition Operation]	1	1%
[Division Operation]	1	1%
[Equals Operation]	1	1%
[Memory Subtract Operation]	1	1%
[Memory Recall Operation]	1	1%
[Memory Save/Add Operation]	1	1%
[Multiplication Operation]	1	1%
[Percent Operation]	1	1%
[Subtraction Operation]	1	1%
Alarm OnOrOff	1	1%
Bangkok [Time Zone]	1	1%
Cairo [Time Zone]	1	1%
Calculator	1	1%
Chicago [Time Zone]	1	1%
Currency Exchange [Calculator]	1	1%
Denver [Time Zone]	1	1%
Hong Kong [Time Zone]	1	1%
Honolulu [Time Zone]	1	1%
Karachi [Time Zone]	1	1%
London [Time Zone]	1	1%
Los Angeles [Time Zone]	1	1%
Moscow [Time Zone]	1	1%
Paris [Time Zone]	1	1%
Rio De Janeiro [Time Zone]	1	1%
Sydney [Time Zone]	1	1%
Tokyo [Time Zone]	1	1%
Wellington [Time Zone]	1	1%

4.5 Morphology

Figure 14 – Protocol Calendar / Calculator Morphological Map

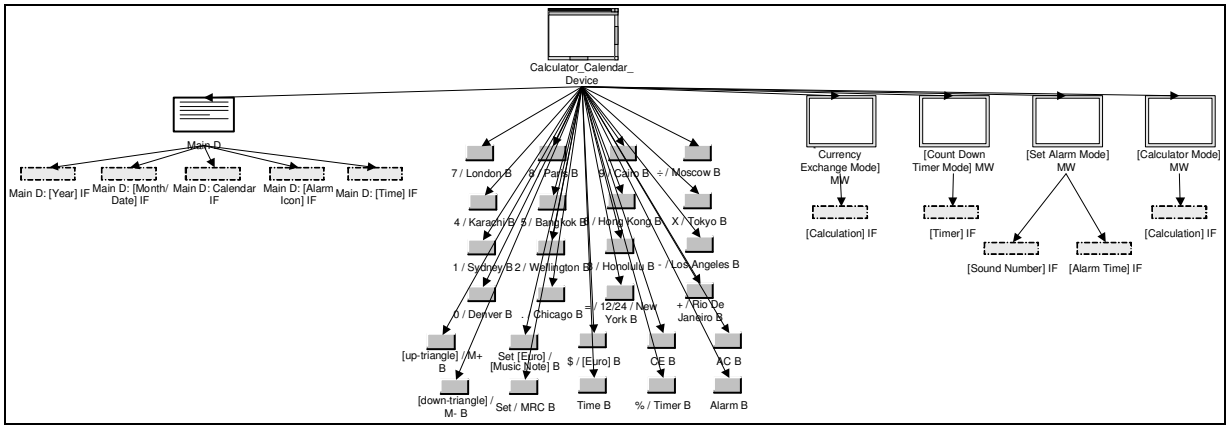


Figure 14 shows the morphological map of the application. Table 41 contains a list of the following morphological elements. They are numbered by the order that they were placed into the diagram.

Table 41 – Protocol Calendar / Calculator Morphological Elements

#	Name
1	Calculator_Calendar_Device
2	Main D
3	Main D: [Year] IF
4	Main D: [Month/Date] IF
5	Main D: Calendar IF
6	Main D: [Time] IF
7	7 / London B
8	8 / Paris B
9	9 / Cairo B
10	÷ / Moscow B
11	4 / Karachi B
12	5 / Bangkok B
13	6 / Hong Kong B
14	X / Tokyo B
15	1 / Sydney B
16	2 / Wellington B
17	3 / Honolulu B
18	- / Los Angeles B
19	0 / Denver B
20	. / Chicago B

21	= / 12/24 / New York B
22	+ / Rio De Janeiro B
23	[up-triangle] / M+ B
24	Set [Euro] / [Music Note] B
25	$ / [Euro] B
26	CE B
27	AC B
28	[down-triangle] / M- B
29	Set / MRC B
30	Time B
31	% / Timer B
32	Alarm B
33	[Calculator / Currency Exchange Mode] MW
34	[Count Down Timer Mode] MW
35	Main D: [Alarm Icon] IF
36	[Set Alarm Mode] MW
37	[Calculation] IF
38	[Timer] IF
39	[Sound Number] IF
40	[Alarm Time] IF

4.6 Ontology

Figure 15 – Protocol Calendar / Calculator Ontology

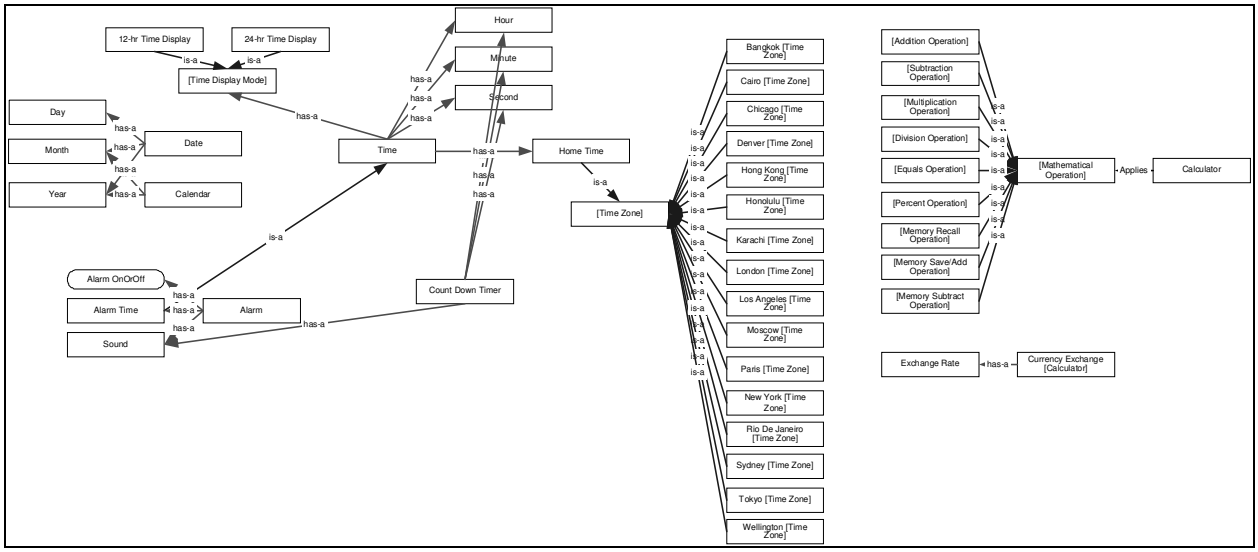


Figure 15 – Protocol Calendar / Calculator Ontology shows the ontology for the application. Table 42 shows a list of the concepts identified in the ontology. They are numbered by the order that they were placed into the diagram and identified as an Entity Type or an Attribute (attributes are concepts that lack independent identity from the connected entity type).

4.6.1 Concepts in Application

Table 42 – Protocol Calendar / Calculator Ontological Elements. The letter E indicates the concept is an Entity Type. The letter A indicates the concept is an Attribute.

#	Concept Name	Type
1	Home Time	E
2	New York [Time Zone]	E
3	12-hr Time Display	E
4	24-hr Time Display	E
5	[Time Display Mode]	E
6	[Addition Operation]	E
7	[Division Operation]	E
8	[Equals Operation]	E
9	[Memory Subtract Operation]	E
10	[Memory Recall Operation]	E
11	[Memory Save/Add Operation]	E
12	[Multiplication Operation]	E
13	[Percent Operation]	E
14	[Subtraction Operation]	E
15	[Mathematical Operation]	E
16	Alarm	E
17	Alarm OnOrOff	A
18	Alarm Time	E

#	Concept Name	Type
19	Bangkok [Time Zone]	E
20	Cairo [Time Zone]	E
21	Calculator	E
22	Chicago [Time Zone]	E
23	Count Down Timer	E
24	Currency Exchange [Calculator]	E
25	Date	E
26	Denver [Time Zone]	E
27	Exchange Rate	E
28	Hong Kong [Time Zone]	E
29	Honolulu [Time Zone]	E
30	Hour	E
31	Karachi [Time Zone]	E
32	London [Time Zone]	E
33	Los Angeles [Time Zone]	E
34	Minute	E
35	Month	E
36	Moscow [Time Zone]	E

#	Concept Name	Type
37	Paris [Time Zone]	E
38	Rio De Janeiro [Time Zone]	E
39	Second	E
40	Sound	E
41	Sydney [Time Zone]	E
42	Time	E

#	Concept Name	Type
43	[Time Zone]	E
44	Tokyo [Time Zone]	E
45	Wellington [Time Zone]	E
46	Year	E
47	Calendar	E
48	Day	E

4.7 Conclusion

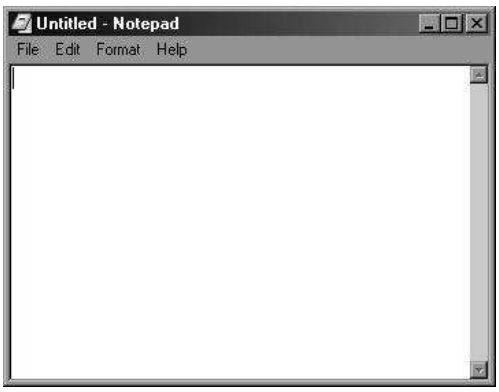
We studied the Protocol Calendar / Calculator for two reasons: to show that our methodologies could be performed on computing applications that were not desktop applications and to have an example of an application with several distinct ontologies encapsulated in a common morphology. As our ontology shows, the Protocol Calendar / Calculator has a Toolbox ontological structure. We were also pleased to see that the ontology has a very low conceptual coherence across all measures but that each subgroup showed a high conceptual coherence. In future versions, we could see developers adding a scheduling system to the device, linking up the Calendar and Time subgroups. The Calculator and Currency Exchange ontologies would still be separate in the ontology but this modification (and changes to the Calculator such as adding more scientific operations) could eventually turn the ontology into an Urban ontological structure where the Scheduling and Calculating concepts compete for space in the limited morphology.

Appendix 5 – Microsoft Notepad Case Study

5.1 Introduction

This appendix describes the results of the ontological excavation, ontological analysis, and use case silhouette analysis of the Microsoft Notepad. MS Notepad (Figure 16) is a text editor that comes with the Windows operating systems. Notepad accepts a variety of types of text files in different encodings and displays and prints a document using application settings that are applied to every text file read by Notepad. These display and print settings do not get saved with the program.

Figure 16 – MS Notepad application



5.2 Modeling Issues

One modeling issue worth noting is our separation of the notions of the Current File containing the text and the Document displayed in the Notepad application itself since they have different properties. Another modeling decision we made concerned the Header / Footer codes and their relationships to items such as Page number. We chose to model the (optional) relationship showing that Headers and Footers can contain the text that describes the File Name, Page Number, Current Time, and so on. We did not link the actual codes to their respective items. We also did two analyses – one including the various scripts that Notepad can interpret (Greek, Arabic, Hebrew, etc) and paper sizes (A4, Legal, Letter, etc) and one without. Concepts discovered late in the analysis (during the Use Case Silhouette) but not included here include: Window and Encoding.

5.3 Ontological Analysis

Below is a summary of the findings from the ontological analysis. They include the following information:

- List of core concepts and their centrality values.
- Subgroups identified by k-core analysis
- Statistics of the ontology

5.3.1 Core Concepts Identified

Core concepts are those concepts essential to that application's ontology. Table 43 shows a list of the concepts identified in the ontology of the application. Values have a range from 0 to 100 where 100 means that the concept has connections to all other concepts in the ontology and 0 means the concept is either an isolate or a leaf node in the ontology. Core concepts have a centrality value greater than or equal to 7.0 and have been italicized.

Table 43 – MS Notepad Concepts ordered by Centrality Value

Concept Name	Value	Description
Page Setup	51.5	The configuration for how a page should be printed.
Font [Setting]	39.6	The settings for the font of the display
Paper	31.7	The medium of printing – settings determine size and source
Text	29.0	A sequence of characters and symbols
Size	25.5	The size of the paper that the document will be printed to
Font	23.8	The typeface, size, and style used to display text in the document.
Script	23.3	The type of script encoding for the current file.
Header	19.4	Text that is found at the top of a page.
Footer	19.4	Text that is found at the bottom of a page.
[Header/Footer Code]	17.6	Codes that determine how text is formatted in the header or footer
[Configuration]	17.0	The settings of the Notepad application that determine how a current file is displayed on screen
Margins	12.0	The intervals between the text and the edge of a page
Alignment	10.5	The setting of text in a margin that determines where it is lined up in a page
Font Style	9.7	The font setting that determines whether character is Regular, Bold, Italic, or Bold Italic
Current File	7.2	The file currently being viewed or edited by the Notepad application
Document	5.0	The current file as displayed and printed by the Notepad application
Source	4.9	The setting of a printer that determines where its paper source will be.
Orientation	4.9	The setting of a page that determines whether it is aligned vertically or horizontally
Line	2.5	A set of characters on a page separated by carriage returns.
Character	2.5	A symbol token.
Date	2.4	Text displaying the month, day, and year
Time	2.4	Text displaying the hour, minute, and second.
File Name	1.5	The name of the current file.
Page Number	1.4	The number designating a page in the document
Word [Token]	1.3	A string of text separated by spaces
Log	0.4	A setting that causes the current date and time to be inserted at the end of a document on opening
Current Date	0.2	The current date according to the computer clock inserted into the document as text
Current Time	0.2	The current time according to the computer clock inserted into the document as text
Left Alignment Code	0.2	A code that formats the text in a header or footer to be aligned to the left side of the page
Right Alignment Code	0.2	A code that formats the text in a header or footer to be aligned to the right side of the page
Center Alignment Code	0.2	A code that formats the text in a header or footer to be aligned in the center of the page
File	0.1	The text in its electronically encoded form
Page	0	A section of the document that can be printed to a piece of paper.
Case	0	The case (lower or upper) of a character
[Font] Sample	0	A set of sample characters displaying a typeface and style
Font Size	0	The height of a character measured in points.
Bold	0	A type of font style
Italic	0	A type of font style
Bold Italic	0	A type of font style
Regular	0	A type of font style
Greek	0	A type of script
Western	0	A type of script
Arabic	0	A type of script

Concept Name	Value	Description
Turkish	0	A type of script
Hebrew	0	A type of script
Baltic	0	A type of script
Central European	0	A type of script
Cyrillic	0	A type of script
Vietnamese	0	A type of script
Mac	0	A type of script
Line Number	0	The number for a line in a document
A4 Small	0	A type of paper
US Letter	0	A type of paper
US Legal Small	0	A type of paper
US Legal	0	A type of paper
US Letter Small	0	A type of paper
A4	0	A type of paper
Com 10 Envelope Center Fed	0	A type of paper
B5	0	A type of paper
Letter	0	A type of paper
Legal	0	A type of paper
Monarch Envelope Center Feed	0	A type of paper
Manual	0	Designates the source of paper as the printer's manual feed slot
Cassette Feed	0	Designates the source of paper as the printer's cassette
Portrait	0	A vertical orientation of a page for printing
Landscape	0	A horizontal orientation of a page for printing
Inches	0	A unit of measurement used to determine margin distances
Left Margin	0	A specified distance between the text and the left edge of the paper.
Right Margin	0	A specified distance between the text and the right edge of the paper.
Top Margin	0	A specified distance between the text and the top edge of the paper.
Bottom Margin	0	A specified distance between the text and the bottom edge of the paper.
File Name Code	0	A code that allows printing the file name in a header or footer
Date Code	0	A code that allows printing the current date in a header or footer
Time Code	0	A code that allows printing the current time in a header or footer
Page Number Code	0	A code that allows printing the page number in a header or footer
Ampersand Code	0	A code that allows printing the '&' symbol in a header or footer
Left Alignment	0	A setting that aligns text on the left side of the page
Right Alignment	0	A setting that aligns text on the right side of the page
Center Alignment	0	A setting that aligns text in the middle of the page
Ampersand	0	The & symbol
Printer	0	The device that prints a document
Extension	0	The text code put on the first line of a document to enable a log.

We performed a second analysis on the ontology (Table 44) without the concepts that are types of Scripts or Size (paper sizes) to see how concepts with many types affected the rest of the ontology. As expected, Script and Size no longer registered as core concepts and the concepts that they had relationships with also decreased in centrality. What was not expected is how little the other concepts were affected by this change.

Table 44 – MS Notepad Concepts ordered by Centrality Value (no Size or Scripts)

Concept Name	Value
Page Setup	45.2
Text	32.5
Font [Setting]	27.9
Header	23.8
Footer	23.8
[Header/Footer Code]	23.2
Margins	16.1
Alignment	13.9
Font Style	13.0
Paper	12.8
[Configuration]	9.3
Current File	9.2
Source	6.6
Orientation	6.6
Document	5.6
Line	3.4
Character	3.3
Font	3.3
Date	3.2
Time	3.2
File Name	2.1
Page Number	2.0
Word [Token]	1.9
Log	0.7
Current Date	0.4
Current Time	0.4
Left Alignment Code	0.3
Right Alignment Code	0.3
Center Alignment Code	0.3
File	0.1
Page	0.1
Case	0

Concept Name	Value
[Font] Sample	0
Font Size	0
Bold	0
Italic	0
Bold Italic	0
Regular	0
Script	0
Line Number	0
Size	0
Manual	0
Cassette Feed	0
Portrait	0
Landscape	0
Inches	0
Left Margin	0
Right Margin	0
Top Margin	0
Bottom Margin	0
File Name Code	0
Date Code	0
Time Code	0
Page Number Code	0
Ampersand Code	0
Left Alignment	0
Right Alignment	0
Center Alignment	0
Ampersand	0
Printer	0
Extension	0

5.3.2 Teleons Identified

Teleons suggest morphological features. We used a *k-core analysis* to identify potential teleons in the ontology. A *k-core* is a connected, maximal, induced subgraph of nodes such that each node has a minimum degree greater than equal to *k*. Teleons identified in the application are listed by their k-value in Table 45 along with the concepts contained in that subgraph. Table 46 shows the analysis without the Size or Script types.

Table 45 – MS Notepad Teleons Identified by K-Core Analysis

k-value	Concepts in Teleon
3	Text, Header, Footer, File Name, Page Number, Date, Time
2	Header / Footer Code, Left / Right / Center Alignment, Alignment of (Header / Footer)
2	File, Current File, [Configuration], Font [Setting], Line, Word [Token], Page Setup, Document, Page, Log, Extension, Current Date, Current Time

Table 46 – MS Notepad Teleons Identified by K-Core Analysis, no Script or Size types (no change to Teleons)

k-value	Concepts in Teleon
3	Text, Header, Footer, File Name, Page Number, Date, Time
2	Header / Footer Code, Left / Right / Center Alignment, Alignment of (Header / Footer)
2	File, Current File, [Configuration], Font [Setting], Line, Word [Token], Page Setup, Document, Page, Log, Extension, Current Date, Current Time

5.3.3 Statistics

The following table (Table 47) lists the overall composition of the ontology. Table 48 has the statistics of the ontology without Script or Size types.

Table 47 – MS Notepad Ontological Metrics

# of entity types:	78
# of attributes:	4
# of nodes in ontology:	82
# of core concepts in ontology:	15
% of total ontology covered by core concepts:	18 %
% of total ontology covered by peripheral concepts:	72 %
% of ontology (no attributes) covered by core concepts:	19 %
% of ontology (no attributes) covered by peripheral concepts:	71 %
Average centrality of concepts	4.48
Density (number of edges divided by total number of possible edges)	.03

Table 48 – MS Notepad Ontological Metrics (no Scripts or Size types)

# of entity types:	57
# of attributes:	4
# of nodes in ontology:	61
# of core concepts in ontology:	12
% of total ontology covered by core concepts:	20 %
% of total ontology covered by peripheral concepts:	80 %
% of ontology (no attributes) covered by core concepts:	21 %
% of ontology (no attributes) covered by peripheral concepts:	79 %
Average centrality of concepts	4.83
Density (number of edges divided by total number of possible edges)	.04

5.4 The Use Case Silhouette

The use case silhouette process takes a set of use cases and uses them to obtain statistics such as the number of concepts present in the ontology and the amount of ontological coverage by those concepts. These findings are summarized in Table 49.

Table 49 – MS Notepad Use Case Silhouette Statistics

Source	Help files of Notepad Application
# of use cases:	32
# concepts invoked:	66
ontological coverage:	80%

5.4.1 Ontological Coverage by Use Case

Table 50 lists the number of concepts activated in each use case and their coverage with respect to the overall ontology. It also shows the proportion of the use case's concepts that are core concepts.

Table 50 – MS Notepad – Use Case Overview

#	Use Case Name	# of Concepts	# of Unique Concepts	% of ontology	% core concepts
1	adding a log	7	7	9%	0%
2	adding page numbers	8	8	10%	63%
3	aligning headers and footers	11	11	13%	36%
4	change margins of a printed document	4	4	5%	25%
5	change page setup	17	11	13%	36%
6	change paper source	5	5	6%	20%
7	change size of paper	3	3	4%	67%
8	changing fonts	10	10	12%	30%
9	character sets	11	11	13%	9%
10	copying text	2	2	2%	100%
11	creating headers and footers	25	25	30%	20%
12	cutting text	2	2	2%	100%
13	deleting text	2	2	2%	100%
14	editing text	2	2	2%	100%
15	find specifc characters or words	3	3	4%	67%
16	finding / going to specific lines	3	3	4%	33%
17	globally replacing text	3	3	4%	67%
18	inserting ampersands	5	5	6%	60%
19	inserting date and time	6	6	7%	33%
20	inserting file names	5	5	6%	60%
21	inserting text	2	2	2%	100%
22	landscape page orientation	4	4	5%	25%
23	moving text	2	2	2%	100%
24	page orientation	5	5	6%	20%
25	pasting text	2	2	2%	100%
26	portrait page orientation	4	4	5%	25%
27	print document	2	2	2%	0%

#	Use Case Name	# of Concepts	# of Unique Concepts	% of ontology	% core concepts
28	printer settings	3	3	4%	0%
29	replacing specific characters or words	3	3	4%	67%
30	samples of fonts, viewing	2	2	2%	50%
31	saving documents	1	1	1%	100%
32	wrap text to window size	2	2	2%	0%

5.4.2 Concept Frequency Across Use Cases

Concept frequency looks at how often a concept is accessed across all the use cases and how often it is accessed against all the concepts invoked by all of the use cases. These are summarized in Table 51. Concept frequency is used to compare against a concept's centrality measures to see whether it retains its importance in the set of use cases. Presumably, a discrepancy would indicate that a concept with structural importance but lacking importance relative to actual usage needs to be made more prominent in the morphology, less prominent in the ontology, or is a symptom of a discontinuity between the system's model of usage and the goals of its users.

Table 51 – MS Notepad Frequency of Concept appearance in use case set. Core concepts are italicized.

Name	# Times Accessed	% of Total # of concepts invoked
Document	16	10%
Text	15	9%
Current File	13	8%
Page Setup	12	7%
[Configuration]	4	2%
Case	4	2%
Orientation	4	2%
[Header/Footer Code]	4	2%
Source	3	2%
Portrait	3	2%
Landscape	3	2%
Printer	3	2%
Current Date	3	2%
Current Time	3	2%
Font [Setting]	2	1%
[Font] Sample	2	1%
Font	2	1%
Script	2	1%
Size	2	1%
Manual	2	1%
Cassette Feed	2	1%
Margins	2	1%
Inches	2	1%
Left Margin	2	1%
Header	2	1%

Name	# Times Accessed	% of Total # of concepts invoked
Footer	2	1%
File Name Code	2	1%
Ampersand Code	2	1%
Alignment	2	1%
Left Alignment Code	2	1%
Right Alignment Code	2	1%
Center Alignment Code	2	1%
Left Alignment	2	1%
Right Alignment	2	1%
Center Alignment	2	1%
Date	2	1%
Time	2	1%
Page Number	2	1%
Font Style	1	1%
Font Size	1	1%
Bold	1	1%
Italic	1	1%
Bold Italic	1	1%
Regular	1	1%
Greek	1	1%
Western	1	1%
Arabic	1	1%
Turkish	1	1%
Hebrew	1	1%
Baltic	1	1%

Name	# Times Accessed	% of Total # of concepts invoked
Central European	1	1%
Cyrillic	1	1%
Vietnamese	1	1%
Line	1	1%
Line Number	1	1%
Word [Token]	1	1%
Paper	1	1%
Right Margin	1	1%
Top Margin	1	1%
Bottom Margin	1	1%
File Name	1	1%
Date Code	1	1%
Time Code	1	1%
Page Number Code	1	1%
Ampersand	1	1%
Extension	1	1%

Name	# Times Accessed	% of Total # of concepts invoked
File	0	0%
Character	0	0%
Mac	0	0%
A4 Small	0	0%
US Letter	0	0%
US Legal Small	0	0%
US Legal	0	0%
US Letter Small	0	0%
A4	0	0%
Com 10 Envelope Center Fed	0	0%
B5	0	0%
Letter	0	0%
Legal	0	0%
Monarch Envelope Center Feed	0	0%
Page	0	0%
Log	0	0%

5.5 Morphology

Figure 17 – A portion of the MS Notepad Morphological Map

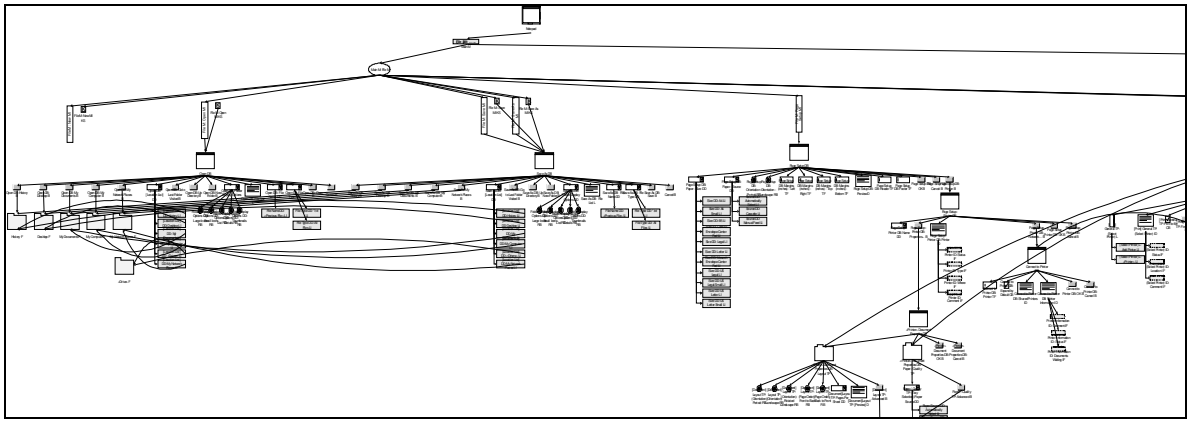


Figure 17 shows a portion of the morphological map of the application. The actual map spans several pages and will be made available outside this document. Table 52 contains a list of the following morphological elements. They are numbered by the order that they were placed into the diagram.

Table 52 – MS Notepad Morphological Elements

#	Name
1	Notepad
2	Main W
3	Main M
4	Main M: File M
5	Main M: Edit M
6	Main M: Format M
7	Main M: Help M
8	File M: New MI
9	File M: Open MI
10	File M: Save MI
11	File M: Save As MI
12	File M: Page Setup MI
13	File M: Print MI
14	Open DB
15	Open DB: History B
16	Open DB: Desktop B
17	Open DB: My Documents B
18	Open DB: My Computer B
19	Open DB: My Network Places B
20	Open DB: [Location List] DD
21	Open DB: Go to Last Folder Visited B
22	Open DB: Up Directory B
23	Open DB: New Folder B

#	Name
24	Open DB: Folder Display Options DD
25	Open DB: File List L
26	Open DB: File Name DD
27	Open DB: File Type DD
28	Open DB: Open B
29	Cancel B
30	History F
31	Desktop F
32	My Documents F
33	My Computer F
34	My Network Places F
35	[Location List] DD: History LI
36	[Location List] DD: Desktop LI
37	[Location List] DD: My Documents LI
38	[Location List] DD: My Computer LI
39	[Location List] DD: <Drives> LI
40	[Location List] DD: My Network Places LI
41	<Drive> F
42	Folder Display Options DD: Large Icons RB
43	Folder Display Options DD: Small Icons RB
44	Folder Display Options DD: List RB
45	Folder Display Options DD: Details RB
46	Folder Display Options DD: Thumbnails RB

#	Name
47	File Name DD: <Previous File> LI
48	File Type DD: *.txt LI
49	File Type DD: All Files LI
50	Save As DB
51	Save As DB: History B
52	Save As DB: Desktop B
53	Save As DB: My Documents B
54	Save As DB: My Computer B
55	Save As: My Network Places B
56	Save As DB: [Location List] DD
57	Save As DB: Go to Last Folder Visited B
58	Save As DB: Up Directory B
59	Save As DB: New Folder B
60	Save As DB: Folder Display Options DD
61	Save As DB: File List L
62	Save As DB: File Name DD
63	Save As DB: File Type DD
64	Save As DB: Save B
65	Cancel B
66	[Location List] DD: History LI
67	[Location List] DD: Desktop LI
68	[Location List] DD: My Documents LI
69	[Location List] DD: My Computer LI
70	[Location List] DD: <Drives> LI
71	[Location List] DD: My Network Places LI
72	Folder Display Options DD: Large Icons RB
73	Folder Display Options DD: Small Icons RB
74	Folder Display Options DD: List RB
75	Folder Display Options DD: Details RB
76	Folder Display Options DD: Thumbnails RB
77	File Name DD: <Previous File> LI
78	File Type DD: *.txt LI
79	File Type DD: All Files LI
80	Page Setup DB
81	Page Setup DB: Paper - Size DD
82	Page Setup DB: Paper - Source DD
83	Page Setup DB: Orientation - Portrait RB
84	Page Setup DB: Orientation - Landscape RB
85	Page Setup DB: Margins (inches) - Left TF
86	Page Setup DB: Margins (inches) - Right TF
87	Page Setup DB: Margins (inches) - Top TF
88	Page Setup DB: Margins (inches) - Bottom TF
89	Page Setup DB: Preview D
90	Page Setup DB: Header TF
91	Page Setup DB: Footer TF

#	Name
92	Page Setup DB: OK B
93	Page Setup DB: Cancel B
94	Page Setup DB: Printer B
95	Size DD: A4 Small LI
96	Size DD: US Letter LI
97	Size DD: US Legal Small LI
98	Size DD: US Legal LI
99	Size DD: Monarch Envelope Center Fed LI
100	Size DD: Letter LI
101	Size DD: Legal LI
102	Size DD: A4 LI
103	Size DD: Com 10 Envelope Center Fed LI
104	Size DD: B5 LI
105	Size DD: US Letter Small LI
106	Source DD: Automatically Select LI
107	Source DD: Cassette LI
108	Source DD: Manual Feed LI
109	Page Setup - Printer DB
110	Page Setup - Printer DB: Name DD
111	Page Setup - Printer DB: Properties... B
112	Page Setup - Printer DB: Printer D
113	Printer ID: Status IF
114	Printer ID: Type IF
115	Printer ID: Where IF
116	Printer ID: Comment IF
117	Page Setup Printer DB: Network… B
118	Page Setup Printer DB: OK B
119	Page Setup Printer DB: Cancel B
120	<Printer> Document Properties DB
121	<Printer> Document Properties DB: Layout TP
122	<Printer> Document Properties DB: Paper / Quality TP
123	<Printer> Document Properties DB: OK B
124	<Printer> Document Properties DB: Cancel B
125	[Document] Layout TP: (Orientation) Portrait RB
126	[Document] Layout TP: (Orientation): Landscape RB
127	[Document] Layout TP: (Orientation): Rotated Landscape RB
128	[Document] Layout TP: (Page Order) - Front to Back RB
129	[Document] Layout TP: (Page Order) - Back to Front RB
130	[Document]Layout TP: Pages Per Sheet DD
131	[Document]Layout TP: [Preview] D
132	[Document] Layout TP: Advanced B
133	Paper / Quality TP: (Tray Selection) Paper Source DD
134	Paper Source DD: Automatically Select LI
135	Paper Source DD: Cassette LI
136	Paper Source DD: Manual Feed LI

#	Name
137	Paper / Quality TP: Advanced B
138	<Printer> Advanced Options DB
139	<Printer> Advanced Options DB: [Printer Options] L
140	[Printer Options List]: Paper / Output LI
141	[Printer Options List]: Graphic LI
142	[Printer Options List]: Document Options LI
143	Paper / Output LI Paper Size DD
144	Paper Output LI: Copy Count LI
145	Graphic LI: Scaling LI
146	Graphic LI: TrueTypeFont [Option] LI
147	Document Options LI: Advanced Printing Features LI
148	Document Options LI: PostScript Options LI
149	PostScript Options LI: PostScript Output Option LI
150	PostScript Options LI: TrueType Font Download Option LI
151	PostScript Options LI: Send PostScript Error Handler LI
152	PostScript Options LI: Compress bitmaps LI
153	PostScript Options LI: Mirrored Output
154	PostScript Options LI: Negative Output LI
155	<Printer> Advanced Options DB: OK B
156	<Printer> Advanced Options DB: Cancel B
157	Size DD: A4 Small LI
158	Size DD: US Letter LI
159	Size DD: US Legal Small LI
160	Size DD: US Legal LI
161	Size DD: Monarch Envelope Center Fed LI
162	Size DD: Letter LI
163	Size DD: Legal LI
164	Size DD: A4 LI
165	Size DD: Com 10 Envelope Center Fed LI
166	Size DD: B5 LI
167	Size DD: US Letter Small LI
168	Connect to Printer DB
169	Connect to Printer DB: Printer TF
170	Connect to Printer DB: Expand by Default CB
171	Connect to Printer DB: Shared Printers ID
172	Connect to Printer DB: Printer Information ID
173	Printer Information ID: Comment IF
174	Printer Information ID: Status IF
175	Printer Information ID: Documents Waiting IF
176	Connect to Printer DB: OK B
177	Connect to Printer DB: Cancel B
178	Print DB
179	Print DB: General TP
180	Print DB: Print B
181	Print DB: Cancel B

#	Name
182	Print DB: Apply B
183	[Print] General TP: (Select Printer) L
184	[Print] General TP: (Select Printer) ID
185	[Print] General TP: Print to file CB
186	[Print] General TP: Find Printer B
187	[Print] General TP: (Page Range) All RB
188	[Print] General TP: (Page Range) Selection RB
189	[Print] General TP: (Page Range) Current Page RB
190	[Print] General TP: (Page Range) Pages RB
191	[Print] General TP: (Page Range) Pages TF
192	[Print] General TP: Number of Copies TF
193	[Print] General TP: Collate CB
194	(Select Printer) ID: Status IF
195	(Select Printer) ID: Location IF
196	(Select Printer) ID: Comment IF
197	(Select Printer) L: Add Printer LI
198	(Select Printer) L: <Printer> LI
199	File M: Exit MI
200	Edit M: Undo MI
201	Edit M: Cut MI
202	Edit M: Copy MI
203	Edit M: Paste MI
204	Edit M: Delete MI
205	Edit M: Find MI
206	Find DB
207	Find DB: Find what TF
208	Find DB: Match case CB
209	Find DB: (Direction) Up RB
210	Find DB: (Direction) Down RB
211	Find DB: Find Next B
212	Find DB: Cancel B
213	Edit M: Find Next MI
214	Edit M: Replace MI
215	Replace DB
216	Replace DB: Find what TF
217	Replace DB: Replace with TF
218	Replace DB: Find Next B
219	Replace DB: Replace B
220	Replace DB: Replace All B
221	Replace DB: Cancel B
222	Replace DB: Match case CB
223	Edit M: Goto MI
224	Goto line DB
225	Goto line DB: [Line #] TF
226	Goto line DB: OK B

#	Name
227	Goto line DB: Cancel B
228	Edit M: Select All MI
229	Edit M: Time/Date MI
230	Format M: Word Wrap MI
231	Format M: Font MI
232	Font DB
233	Font DB: Font L
234	Font DB: Font Style L
235	Font DB: Size L
236	Font DB: Sample D
237	Font DB: Script DD
238	Font DB: OK B
239	Font DB: Cancel B
240	Font L: <Font> LI
241	Font Style L: Regular LI
242	Font Style L: Italic LI
243	Font Style L: Bold LI
244	Font Style L: Bold Italic LI
245	Size L: [Point Size] LI
246	Script DD: Western LI
247	Script DD: Greek LI
248	Help M: Help Topics MI
249	Help M: About Notepad MI
250	Main W: [Text]
251	Main W: VSB

#	Name
252	File M: New MI KS
253	File M: Open MI KS
254	File M:: Save MI KS
255	File M: Save As MI KS
256	File M: Print MI KS
257	Edit M: Undo MI KS
258	Edit M: Cut MI KS
259	Edit M: Copy MI KS
260	Edit M: Paste MI KS
261	Edit M: Delete MI KS
262	Edit M: Find All MI KS
263	Edit M: Find Next MI KS
264	Edit M: Replace MI KS
265	Edit M: Goto MI KS
266	Edit M: Select All MI KS
267	Edit M: Time / Date KS
268	Script DD: Hebrew LI
269	Script DD: Arabic LI
270	Script DD: Turkish LI
271	Script DD: Baltic LI
272	Script DD: Central European LI
273	Script DD: Cyrillic LI
274	Script DD: Vietnamese LI
275	Script DD: Mac LI
276	Main W: HSB

5.6 Ontology

Figure 18 – MS Notepad Ontology

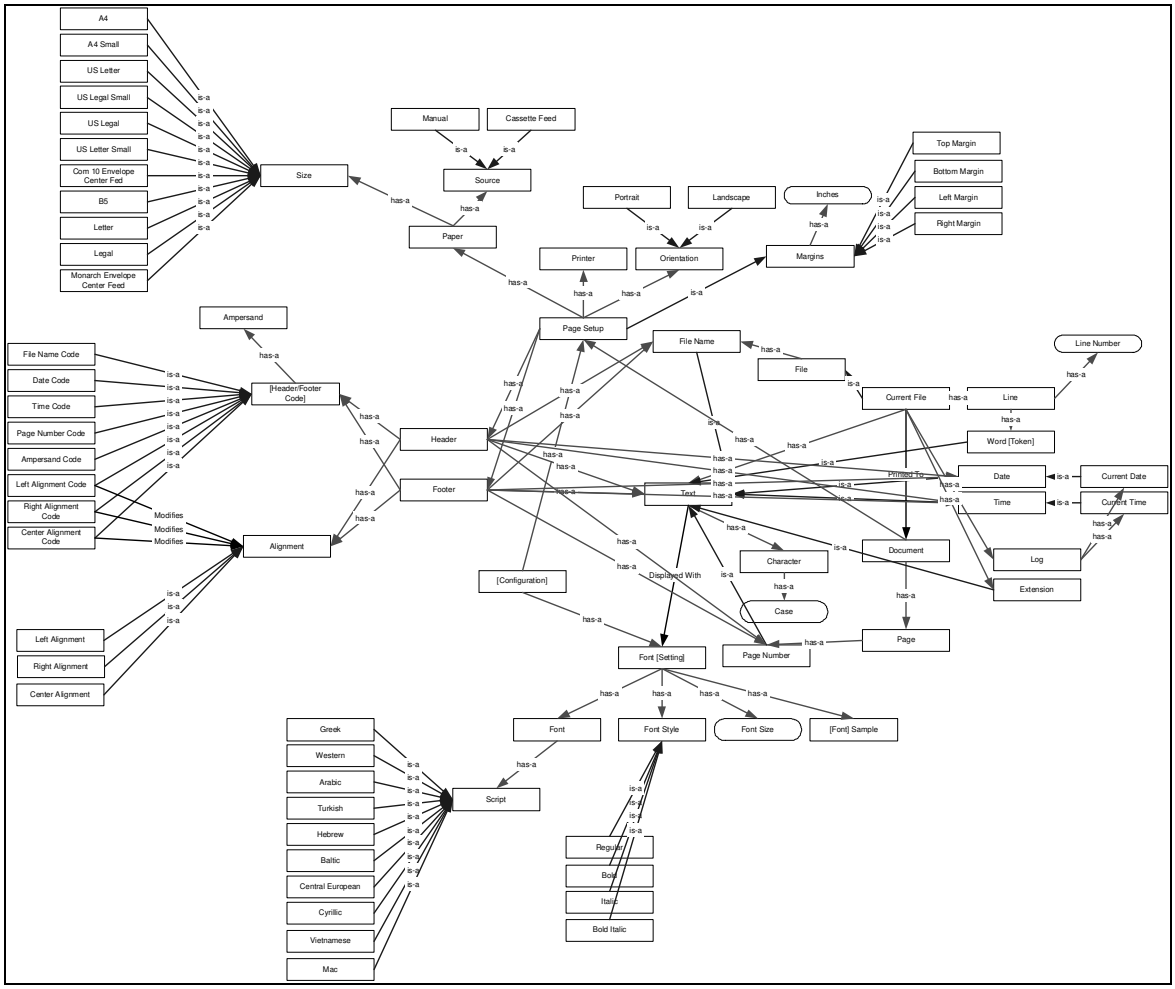


Figure 18 shows the ontology for the application. Table 53 shows a list of the concepts identified in the ontology. They are numbered by the order that they were placed into the diagram and identified as an Entity Type or an Attribute (attributes are concepts that lack independent identity from the connected entity type).

5.6.1 Concepts in Application

Table 53 – MS Notepad Ontological Elements. The letter E indicates the concept is an Entity Type. The letter A indicates the concept is an Attribute.

#	Concept Name	Type
1	File	E
2	Current File	E
3	[Configuration]	E
4	Text	E
5	Character	E
6	Case	A
7	Font Style	E

#	Concept Name	Type
8	Font [Setting]	E
9	[Font] Sample	E
10	Font Size	A
11	Bold	E
12	Italic	E
13	Bold Italic	E
14	Regular	E

#	Concept Name	Type
15	Font	E
16	Script	E
17	Greek	E
18	Western	E
19	Arabic	E
20	Turkish	E
21	Hebrew	E
22	Baltic	E
23	Central European	E
24	Cyrillic	E
25	Vietnamese	E
26	Mac	E
27	Line	E
28	Line Number	A
29	Word [Token]	E
30	Page Setup	E
31	Paper	E
32	Size	E
33	A4 Small	E
34	US Letter	E
35	US Legal Small	E
36	US Legal	E
37	US Letter Small	E
38	A4	E
39	Com 10 Envelope Center Fed	E
40	B5	E
41	Letter	E
42	Legal	E
43	Monarch Envelope Center Feed	E
44	Source	E
45	Manual	E
46	Cassette Feed	E
47	Orientation	E
48	Portrait	E
49	Landscape	E

#	Concept Name	Type
50	Margins	E
51	Inches	A
52	Left Margin	E
53	Right Margin	E
54	Top Margin	E
55	Bottom Margin	E
56	Header	E
57	Footer	E
58	[Header/Footer Code]	E
59	File Name	E
60	File Name Code	E
61	Date Code	E
62	Time Code	E
63	Page Number Code	E
64	Ampersand Code	E
65	Alignment	E
66	Left Alignment Code	E
67	Right Alignment Code	E
68	Center Alignment Code	E
69	Left Alignment	E
70	Right Alignment	E
71	Center Alignment	E
72	Date	E
73	Time	E
74	Document	E
75	Page	E
76	Page Number	E
77	Ampersand	E
78	Printer	E
79	Log	E
80	Extension	E
81	Current Date	E
82	Current Time	E

5.7 Conclusion

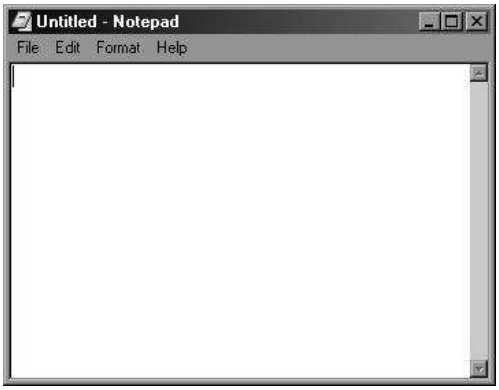
We believed Notepad to be ontologically equivalent to *vi* on UNIX systems and were surprised when our excavation revealed a number of concepts concerning the visual appearance of the text in the application and how the document will look on printing. The teleons in the ontology show three basic features: text editing and management, file handling and application configuration, and header / footer management. While we know that developers probably consider these major groups to be composed of multiple features, we believe that there is evidence that suggests the usefulness of teleons in identifying major groups of functionality in an application. We believe Notepad has an Urban ontological structure where the text processing features are competing with the features that support the viewing and printing configuration of the file. This is somewhat confirmed by our use case silhouettes which showed that many of the use cases with high ontological coverage concern these secondary functions.

Appendix 6 – MS Notepad Case Study 2

6.1 Introduction

This appendix describes the results of the ontological excavation, ontological analysis, and use case silhouette analysis of the Microsoft Notepad (v. 5). MS Notepad (Figure 19) is a text editor that comes with the Windows operating systems. This is a refinement of the original case study (Appendix 5) with modifications to the modeling.

Figure 19 – MS Notepad application



6.2 Modeling Issues

The following changes were made from the previous analysis to enhance the correctness of the model with respect to traditional modeling conventions and to test assumptions about core concepts and conceptual subgroups and sensitivity of our analysis to modeling. Some of these changes resulted after reflection on the 'black box' methods used to obtain the information, producing refinement to the procedures. The following changes were made:

- Proper names were removed from the analysis. This included entity types such as types of Scripts and Paper Sizes which were declared to be instances of their parent class.
- Entity types that did not have an independent identity were refined into attributes of their parent class. For example, in Font Style, we had modeled Regular, Italic, Bold, and Bold Italic as their own entity types. However, these do not have any meaning except in the context of the setting of the display fonts. We turned these into attributes that expressed state (e.g. RegularOrItalicOrBoldOrBoldItalic).
- Current Date and Current Time were collapsed into Date and Time as Notepad only inserts the most current date and time. Any other date and time in the document is simply considered to be part of the text.

The following differences were identified in the new model from the old model:

- Most of the core concepts remained but reordered themselves. For example, Text moved from 4^{th} to 2^{nd} place.
- Font Style, Size, and Script dropped out of the core concept set.
- Source (source of the paper for printing) with two types of sources was added to the list of core concepts.
- [Header / Footer Code] moved up significantly on the list.

6.2.1 Core Concepts Identified

Table 54 shows a list of the core concepts (and some of the concepts under the cutoff point) identified in the ontology of MS Notepad in the original and second versions.

Table 54 – MS Notepad Concepts ordered by Centrality Value. Version 1 is on the left, Version 2 is on the right.

Concept Name (v.1)	Value
Page Setup	51.5
Font [Setting]	39.6
Paper	31.7
Text	29.0
Size	25.5
Font	23.8
Script	23.3
Header	19.4
Footer	19.4
[Header/Footer Code]	17.6
[Configuration]	17.0
Margins	12.0
Alignment	10.5
Font Style	9.7
Current File	7.2
Document	5.0
Source	5.0
Orientation	5.0
Line	2.5
Character	2.5
Date	2.4
Time	2.4
File Name	1.5
Page Number	1.4
Word [Token]	1.3
Log	0.4
Current Date	0.2
Current Time	0.2
Left Alignment Code	0.2
Right Alignment Code	0.2
Center Alignment Code	0.2
File	0.1
Page	0

Concept Name (v.2)	Value
Page Setup	41.7
Text	35.2
[Header/Footer Code]	27.7
Font [Setting]	26.7
Header	23.9
Footer	23.9
Paper	15.3
Alignment	8.9
Current File	8.7
[Configuration]	8.2
Font	7.9
Source	7.9
Margins	7.9
Document	5.4
Line	4.1
Character	4.0
Font Style	4.0
Orientation	4.0
File Name	2.5
Page Number	2.4
Word [Token]	2.3
Current Date	1.4
Current Time	1.4
Log	0.3
Left Alignment Code	0.2
Right Alignment Code	0.2
Center Alignment Code	0.2
File	0.1
Page	0.1

6.2.2 Teleons Identified

There were virtually no changes to the teleons identified in Version 1 (Table 55) from Version 2 (Table 56). Date and Time became Current Date and Current Time in the new version.

Table 55 – MS Notepad Teleons Identified by K-Core Analysis – Version 1

k-value	Concepts in Teleon
3	Text, Header, Footer, File Name, Page Number, Date, Time
2	Header / Footer Code, Left / Right / Center Alignment, Alignment of (Header / Footer)
2	File, Current File, [Configuration], Font [Setting], Line, Word [Token], Page Setup, Document, Page, Log, Extension, Current Date, Current Time

Table 56 – MS Notepad Teleons Identified by K-Core Analysis – Version 2

k-value	Concepts in Teleon
3	Text, Header, Footer, File Name, Page Number, Current Date, Current Time
2	Header / Footer Code, Left / Right / Center Alignment, Alignment of (Header / Footer)
2	File, Current File, [Configuration], Font [Setting], Line, Word [Token], Page Setup, Document, Page, Log, Extension

6.2.3 Statistics

The following tables (Table 57 and Table 58) show an overview of the two ontologies.

Table 57 – MS Notepad Ontological Metrics – Version 1

# of entity types:	78
# of attributes:	4
# of nodes in ontology:	82
# of core concepts in ontology:	15
% of total ontology covered by core concepts:	18 %
% of total ontology covered by peripheral concepts:	72 %
% of ontology (no attributes) covered by core concepts:	19 %
% of ontology (no attributes) covered by peripheral concepts:	71 %
Average centrality of concepts	4.48
Density (number of edges divided by total number of possible edges	.03

Table 58 – MS Notepad Ontological Metrics – Version 2

# of entity types:	51
# of attributes:	10
# of nodes in ontology:	61
# of core concepts in ontology:	13
% of total ontology covered by core concepts:	21 %
% of total ontology covered by peripheral concepts:	79 %
% of ontology (no attributes) covered by core concepts:	25 %
% of ontology (no attributes) covered by peripheral concepts:	75 %
Average centrality of concepts	5.42
Density (number of edges divided by total number of possible edges	.03

6.2.4 Conclusion

Core concepts and teleons seem to be fairly robust to large changes to the ontology (removing or redefining 27 nodes). Virtually no changes were made to the teleons identified. Most of the same basic concepts remained the same with some dropping out of the analysis as they lost their connecting nodes. Some of the ordering of the core concepts also changed after the modeling modification, arguably showing a cleaner analysis of Notepad. Both ontologies show Notepad to be a text editor but one primarily structured around the presentation and printing of text. A further refinement could also remove the printing functions from Notepad's ontology as these are functions common to all Windows applications. This would remove items such as Paper, Source, Manual, and Cassette Feed. However, we are fairly confident that this refinement would have a negligible effect on the current list.