Idris Hsi

Research Statement

Introduction

In the world of commercial computing, we are being inundated with baroque, bloated, and difficult to use software applications. Market forces demand that new versions of software have more features than its predecessors or competitors. However, additional features often produce interaction problems with existing features and decrease the usability of the system. Ensuring that planned features required for market competitiveness enhance the software without these side effects first requires that we understand how software features contribute to the overall design and *conceptual integrity* of the application. Conceptual integrity is the single most important consideration in the effective design of software. Researchers in software engineering have identified symptoms of low conceptual integrity in applications, such as high coupling in the software architecture, “bad smells” in code, and bloat in software features. These symptoms are found in software that is difficult to understand, hard to maintain, and hard to use. However, since the significance of conceptual integrity was defined by Fred Brooks in *The Mythical-Man Month*, little progress has been made on developing models and methods for measuring it, engineering it, or teaching it. While researchers have studied design for specific aspects of software systems, I believe conceptual integrity must be treated as a holistic quality to be designed into all levels of a software system, which can only result, as Brooks says, from a unified design vision. I want to develop a methodology for designing applications from this unified perspective. A methodology that allowed developers to incorporate conceptual integrity into an integrated design that encompasses interfaces, functionality, and software architecture would help bring software engineering one step closer to becoming a mature discipline. As the first step towards this goal, I developed both a research framework and specific methodologies in my PhD thesis work that explain how conceptual integrity manifests itself in software and how it can be quantified.

Conceptual Integrity

While conceptual integrity should be present in all aspects of software, I chose specifically to examine how an application exhibits this integrity in its user-visible services and behaviors. Software is designed and constructed to solve problems in the world. For example, a spreadsheet application embodies a theory of how its users want to organize and modify data; a financial management system embodies a theory about how someone keeps track of their bank accounts. Conceptual integrity begins with problem concepts that were engineered into the code. If those concepts lack integrity then the resulting software is difficult to understand, difficult to use, or does not succeed in helping its users to achieve their goals. In my PhD dissertation, I identify and investigate the quality of *conceptual coherence*, the degree to which the concepts in an application are related to one another, that can serve as a first but crucial approximation for the conceptual integrity of an application. Applications that are perceived as bloated have a disproportionate number of features that go unused by their users. These unused features implement concepts that have poor relationships with the central concepts in the application. In a conceptually coherent application, one expects a set of concepts in an application to have relationships with one another such that all the concepts describe the problems and the solutions the application was engineered to solve. An application with low conceptual coherence contains concepts that are loosely related or unrelated to those central problems and solutions. Because these extraneous concepts appear in services that are accessed through the user interface, they not only occupy valuable screen real estate but require extra effort to learn and understand. I argue that low conceptual coherence reflects a low conceptual integrity in the design of the application. But how does adding a feature affect this coherence? Is there a ‘breakpoint’ where adding a feature causes the application to become ‘bloated’? Can adding features reduce the overall usefulness of an application relative to a specific set of users and work context? Because qualities like bloat or usefulness vary from user to user and context to context, both objective and subjective methods are needed to analyze conceptual coherence of applications and the impact of this coherence on its usefulness relative to specific work contexts.

Ontological Excavation and Analysis

First, measuring conceptual coherence requires that we identify those concepts that are visible to the users of the application in an objective fashion. To identify these user-visible concepts, I developed *ontological excavation* which uses black-box reverse engineering on the application user interface and services. White-box reverse engineering techniques analyze code directly. However, source code often contains concepts that have to do with managing computing services such as memory, file management, and hardware interactions: concepts that are not typically visible to the user. Black-box techniques examine application's behaviors and ignore the source code that instrument these behaviors. First, this method develops a model of the user interface of the application being studied. Nouns and descriptors are identified in the labels of the user interface elements. Then, using data modeling techniques that are also applied in object-oriented programming and database design in combination with observing system interactions and reading help files, these nouns and observed behaviors are refined into a semantic network of concepts and relationships that models the *ontology* of the application. Because a semantic network is a graph consisting of nodes that are concepts and edges that are relationships, graph theoretic techniques from social network analysis can be used to analyze it – specifically, *centrality measures* that can measure the structural importance of concepts in the ontology. Concepts that are highly central in the ontology are *core concepts*, concepts essential to the definition of that application (e.g. removing the concept of 'cell' from a spreadsheet ontology or 'paragraph' from a word processor fundamentally changes or "breaks" those applications). These measures produced candidate metrics that could be used to measure the conceptual coherence of an application.

Empirical Studies

To test these methodologies and metrics, I conducted empirical studies on real applications, such as the Windows CD Player and Microsoft Notepad. Those case studies showed, first, that the approach was tolerant of small biases in data modeling, second, that the core concepts of an application can be identified mathematically, and, lastly, that it is possible to measure the conceptual coherence of an application. I also study several larger applications, including Microsoft Word and Yahoo's Instant Messenger, to demonstrate scalability of the techniques. I also test the claim that conceptual integrity affects the *usefulness* of an application. I 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. I argue that an application with low conceptual coherence has a disproportionate number of concepts that do not contribute to the overall use of the application. To measure this subjective potential use of an application relative to a specific context, I developed a technique called *use case silhouetting* that measures the number of unique concepts that are invoked by a set of task scenarios, selected by actual use. If a set of scenarios invokes most of the concepts in an application ontology, then that set has a high *ontological coverage* and shows that the application is likely to be useful to users who employ those scenarios. In my dissertation, I excavate the ontology of Microsoft Word 2000 and measure its ontological coverage using independently obtained usability data to show that a correlation exists between Word's low conceptual coherence and the low ontological coverage of a set of commonly used features, which indicates that Word's lack of conceptual integrity in its features contributes to user perceptions of its bloat and complexity.

Future Work

My immediate research following the completion of my degree will be to explore applications of my ideas to develop visualizations of software from its features, to correlate interface structure with the centrality of concepts in its ontology, and to develop techniques for ensuring that applications have conceptual fitness with the problem domains of their users. However, my ultimate research objective is to develop a design methodology that ensures conceptual integrity throughout all aspects of software. It is my feeling that as software engineering matures as a discipline and reusable libraries and components becomes a common technology, we will need a design practice that develops the computing models and blueprints that convey both intent, form, and functionality to customers and developers. These design artifacts for computing would be used in the same way that architects use drawings, blueprints, and architectural models in their practice. With such artifacts, designers of computing applications could focus on understanding the problem domain and identifying the necessary services, information models, and interaction techniques while engineers could focus on implementation, ensuring a unified vision throughout development and an appropriate separation of concerns. Many research questions will need to be solved before such a design discipline can be realized. Their answers will require a wide range of scientific and engineering approaches across a number of domains. For example, are there existing systems that exemplify high conceptual integrity that can be identified, for example, from open source communities? What should the proper relationship between ontology and software architecture be to facilitate both design and engineering? Do applications have limits to the number of concepts they can contain relative to their features and their user interface structures? Can applications be designed to accommodate changing requirements and user needs over time? These are all important and interesting questions that I will address in my future research.