From Software Reuse to Example-Based Design

Scott Henninger

Department of Computer Science & Engineering

115 Ferguson Hall, 880115

University of Nebraska

Lincoln, NE 68588-0115

Tel: (402) 472-8394

Email: scotth@cse.unl.edu

Fax: (402) 472-7767

Abstract

In this position paper, I address three separate issues. The first has to do with an overly restricted scope in current software reuse work. Example-based design refers to a superset of software reuse in which components are viewed as examples that can be used to explore previous design solutions and provide a concrete context for learning system concepts, in addition to providing material for software reuse. The second issue is that we need ways of better understanding the process of using examples as a basis for design. A preliminary process model is presented and ways of incorporating example-based design into the design process are suggested. Third, repositories and tools for relevant information is an important element in example-based design. Shortcomings of current software retrieval systems are outlined, and solutions are briefly presented.

Key Words: software reuse, reuse environments, software design, reuse process retrieval, software repository.

Workshop Goals: Learn more about reuse in corporate settings, refine my views on example-based design, increase the awareness and knowledge base on cognitive barriers to reuse.

Working Groups: tools and environments, reuse process models, domain analysis/engineering, reuse and OO methods

1 Background

My involvement in the software reuse field has focused on the development of tools to aid the process of software reuse. My dissertation work involved the construction and evaluation of a retrieval tool, named CodeFinder, that combined a connectionist-based retrieval model with tools for iterative query construction to support retrieval and exploration of repositories. A series of empirical studies was conducted on a repository consisting of over 1800 components. This study revealed that the re-use of source code was only one way that subjects made use of the repository. Subjects also searched for specific functionality to understand how existing features worked, extracted design information that was used to create code from scratch, and located parts where their code needed to fit into the existing system. The components therefore served as concrete examples that facilitated the development process. These findings have led me to begin broadening the view of software reuse and repository tools to include these example-based techniques, and explore other ways in which examples can improve the design and development process.

2 Position

The traditional definition of software reuse has centered on the notion that a software object is reused in a new system [5,6]. This makes sense, as the objective is to use the results of previous design efforts, but it also takes an overly restrictive view of how existing software artifacts can facilitate the design process. Take, for example, a situation in which a designer needs a technique that arranges a listing of e-mail messages so that replies to previous messages are indented. The designer uses a network news reader that performs just this function for news messages. Looking at the source code for the news reader, the designer finds out how messages are arranged and indented, but the environment for the news reader differs enough from the e-mail system that the code cannot be re-used. The designer goes back to the e-mail application and designs the message arrangement facility from scratch, but with an improved understanding of how it should be done.

Has the designer engaged in software reuse? By most traditional definitions of reuse, no. Software reuse must re-use the software. But if we reformulate the question to ask whether the design process has been facilitated, then we must believe that something good has happened. The designer consulted an existing example that transformed an ill-defined design task into a straightforward coding task. The traditional definition of software reuse cannot easily accommodate activities that do not involve physical reuse. I believe that repositories of existing code can support a much wider range of activities. For example, a working piece of code can be used to explore previous design solutions and can provide a concrete context for learning system concepts in addition to providing material for software reuse. For these reasons, I assert that the software reuse community needs to broaden its scope and pursue an *example-based design* methodology for its research context. As I define it, example-based design is the process of using existing examples of design artifacts to facilitate the design process, where the re-use of existing design artifacts is one of many possibilities.

2.1 Technical Issues for Example-Based Design

The current wisdom in software reuse research and practice has been leaning toward the position that managerial issues are the dominant barrier to software reuse. This position asserts that current reuse methods and tools are adequate. It is my conviction that while managerial and incentive issues need to be more fully explored, significant cognitive barriers to the reuse of code remain largely unexplored and unresolved. Current technical solutions to these problems have not yet reached the critical point where the effort involved in re-using code or other design artifacts (such as specifications) is equal to or less than creating design artifacts from scratch. Software reuse and example-based design does not come for free, and we need to build a better understanding of the costs of building and using repositories of code and other design artifacts.

To understand these costs, we must first seek to develop models of example-based design. Development of such models will facilitate an understanding of how the design process is impacted when examples are an integral part of the design process. Figure 1 is a starting point toward this understanding [2]. It identifies three cognitive processes involved

Figure 1: The Process of Using Examples in Design.

in example-based design and their relationships. This diagram in not intended as a comprehensive model of software development, but as a stepping stone to the development of a model of example-based design. While more comprehensive models have been developed [4], this model provides a flavor for the kinds of models needed to advance an understanding of the example-based design process.

This figure also underscores the need for process models and development methodologies that incorporate reuse and exploring the corporate repository for ideas and code. We need models of how existing systems can be used as a *basis* for new systems, not just in terms of re-using existing source code, but in terms of using and building on ideas and features of previous systems. Designers need to see existing systems as their first option, and programming as a last resort. From a systems design standpoint, such a development methodology would greatly benefit from information about how successful different features in previous systems were. Was the feature used in its intended manner? Did users report trouble using it? Feedback of this kind is an important ingredient for the evolution of better software systems.

2.2 Retrieval Tools for Example-Based Design

Another limiting element of the software reuse perspective concerns restricting library components to those meeting reusability certification requirements. The certification process is an excellent means for developing quality reusable code, but the costs of developing code and deigns that transcend the current development project have proven to be quite high, perhaps prohibitively so.[1] Intermediate levels are needed that involve lower costs, while delivering relevant examples of working code, regardless of whether it meets reusability standards. Because example-based design is concerned more with using examples to facilitates design, and not just re-using artifacts, the certification constraint can be relaxed to allow levels of certification and non-certified components into the repository, with appropriate labeling to inform designers of a component's certification properties.

Lifting the ban on non-certified components places an extra burden on the repository. Finding relevant examples in large repositories requires sophisticated retrieval tools. Current techniques for retrieving software components for reuse have failed in two important ways: First, most techniques have focused on elaborate retrieval models that require information to be structured in sophisticated ways. Second, retrieval tools have been based on the assumption that users have a well-defined retrieval need and can easily construct an appropriate query.

The construction of a repository and the method used to retrieve objects in it are closely related. For example, a retrieval system that uses hierarchical categorization as its basis for retrieval must use a repository that is organized hierarchically. The costs and benefits of the requisite information structure must be weighed carefully before choosing a retrieval method. Choosing a method that requires a highly structured repository will dramatically increase the costs of building a repository. Lower cost methods are needed that

[1] Some would claim that building *any* software system is prohibitively high, much less building reusable components while building a system.

allow repositories to be constructed at low cost and combined with intelligent retrieval methods that can make use of low-structure repositories.

Retrieval needs for software designers span a continuum from well-defined look-up ("what's the parameter structure for *sort* in Common Lisp?") to ill-defined design problem ("how would one go about designing an intelligent e-mail reply facility?") [4]. Well-defined problems can be easily solved with current database and string matching technologies. Finding relevant information for ill-defined problems is as much a matter of supporting the problem solving process as it is retrieval mechanics. The retrieval process in these cases is a learning process, where the users learn about the structure and contents of the repository and refine their information need as they query and browse the information space. This indicates the crucial need for flexible querying methods that allow the iterative refinement of queries.

To this end, I have developed a software retrieval tool named CodeFinder that couples a retrieval model relying on very little a-priori structuring with a semi-intelligent tool for extracting components and key terms from existing source code files [3,4]. Codefinder uses a connectionist-based retrieval method that can induce relationships among repository objects without the need for elaborate repository structuring. Codefinder supports iterative refinement of information needs through a technique called retrieval by reformulation [7], which provides methods for incrementally defining queries and browsing the repository. Although empirical results have shown this method to be adequate, CodeFinder also provides facilities that can help users incrementally construct a more structured repository. These re-structuring tools take effect in the process of finding relevant information so that the resulting structures evolve in the context of use as opposed to an arbitrary categorization created by a repository administrator that may or may not reflect the needs of the users.

3 Comparison

The concept of example-based design is complementary to much of the software reuse literature, which has focused exclusively on re-using source code. Some have begun to research how specifications and other design artifacts can be re-used, but none have explored alternative ways in which a repository of examples can be used to facilitate the design process. This position paper also augments current research in certifying reusable components [1] by acknowledging that a continuum of certification can exist in an example-based design strategy.

References

[1] G. Caldira, V.R. Basili, "Identifying and Qualifying Reusable Software Components", *IEEE Computer,* 24(2), Feb. 1991, pp. 61-70.

[2] G. Fischer, S.R. Henninger, D.F. Redmiles, "Cognitive Tools for Locating and Comprehending Software Objects for Reuse", *Thirteenth International Conference on Software Engineering* (Austin, TX), ACM, IEEE, Los Alamitos, CA, 1991, pp. 318-328.

[3] S. Henninger, "Retrieving Software Objects in an Example-Based Programming Environment", *Proceedings SIGIR '91* (Chicago, IL), ACM, 1991, pp. 251-260.

[4] S.R. Henninger, *Locating Relevant Examples for Example-Based Software Design*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, 1993.

[5] R. Prieto-Diaz, "Implementing Faceted Classification for Software Reuse", *Communications of the ACM*, 35(5), May 1991.

[6] T.A. Standish, "An Essay on Software Reuse", *IEEE Transactions on Software Engineering*, SE-10(5), Sept. 1984, pp. 494-497.

[7] M.D. Williams, "What Makes RABBIT Run?", *International Journal of Man-Machine Studies,* 21, 1984, pp. 333-352.

4 Biography

Scott Henninger is an Assistant Professor in the Department of Computer Science and Engineering at the University of Nebraska. He received a BSEE from the University of Southern California in 1983. The next five years were spent as a development engineer working on flight control systems and mainframe storage subsystems. He then returned to academics, earning a Ph.D. in Computer Science, with a certificate in Cognitive Science, from the University of Colorado in May, 1993. His dissertation was on retrieval tools for software reuse repositories, and his current interests, all from a human factors standpoint, are in software reuse environments, requirement and specification technologies, design issues for multi-media systems, and cognitive, organizational, and social issues in the design and development of software.