Reusing Software to Produce Broadband Services

L. F. Capretz and M. A. M. Capretz

University of Aizu
Department of Computer Software
Aizu Wakamatsu, Fukushima, 965-80 JAPAN
lcapretz@u-aizu.ac.jp
mcapretz@u-aizu.ac.jp

ABSTRACT

Software reusability is a technique for improving productivity and quality, which is finally finding general acceptance. A model for software reuse is described as part of a software development life cycle. The approach takes into account software development with reuse of existing components as well as production of assets for future reuse. A successful experience using this framework to produce broadband services is also presented.

INTRODUCTION

Software reuse can be broadly defined as the use of engineering knowledge or assets from existing software systems to build new ones. This is a natural technique for improving software quality and productivity. Individuals and small groups have always practiced *ad hoc* software reuse. Now many organizations want to achieve systematic reuse, which is application domain focused, based on a controlled process model, and concerned primarily with the reuse of higher level life cycle components, such as requirements, designs and frameworks [1].

Another idea which has been receiving a great deal of attention from software engineers is the object-oriented paradigm. Because of the perceived importance of an object-oriented approach, several methodologies have recently emerged to support object-oriented software production [2, 3, 4]. One of the main aspects from this paradigm that seduces software engineers is the possibility to reuse software.

The concepts of reusability and the object-oriented paradigm are so interrelated that it is often difficult to talk about one without mentioning the other. A key concept in software reuse is the application domain, which may be defined as an application area or, more formally, a set of software systems that share similar design decisions. Systematic software reuse thus is a paradigm shift in software engineering from building a single software system to building families of related software systems. The goal of research in software reuse is to create techniques for engineering new software systems from existing software assets (components) [5, 6].

In Europe, a group of companies, universities and research centers from several countries have gathered themselves and taken up the challenge of improving software development productivity within the telecommunications systems industry. Together with the Commission of the European Communities (CEC), they have sponsored a programme called RACE (Research into Advanced Communications in Europe); and in particular, the BOOST project (Broadband Object-Oriented Service Technology) [7, 8]. BOOST is a software environment to develop advanced telecommunications service engineering software [9, 10]. Among its aims is to encourage software reusability within the telecommunications software industry.

The next sections discuss a software life cycle model, named the X model, for software reuse applicable while the software development process is being carried out. The model focuses on a collection of components that can be taken from reusable libraries as well as the production of potential reusable components. Such a model addresses the mechanisms used when components are taken from and stored into reusable libraries. Moreover, the X model recognizes the iterative nature of software development, hence iterations are naturally incorporated into the process where appropriate.

0-7803-3019-6/96/$3.00 © 1996 IEEE.

THE BOOST ENVIRONMENT

Within the scope of software development for the telecommunications application domain, the definition of a CASE environment that makes easy the introduction of new network services is of paramount importance. Such a platform should be able to support as many of the existing software engineering tools as possible in order to extend existing technologies to the task of defining new services. Additionally, the introduction of new services must be fast (mainly in the multimedia arena), thus there is a growing need of reusing already available service engineering functional components.

As broadband systems evolve, existing infrastructures are getting a new lease on life through Integrated Service Digital Network (ISDN). Since 1988, AT&T has deployed wideband ISDN – which can deliver high-quality color images simultaneously with voice and data – at more than three hundred locations in the United States, and in a dozen countries abroad. In a little more than three years, seventy percent of all access lines in the United States will be ISDN-capable.

Across the North Atlantic, some European companies formed a consortium to finance the BOOST project to face the challenge of rapidly creating broadband services. In order to equip the service engineering industry to meet the market needs, BOOST aims at providing a CASE environment for service engineering by taking a pragmatic way of enhancing available technologies and evaluating them in a series of usage trials.

The BOOST partners and their origin are presented in Table 1.

Specifically, the major objectives of the BOOST project are:

- To deliver an object-oriented service engineering environment, based on the rapid enhancement of existing software engineering tools.
- To evaluate and demonstrate the environment through a series of trials.
- To ensure the early availability of service engineering tools for use in some application pilots and other interested RACE projects.
- To ensure the uptake of the environment by the telecommunications services industry.

The BOOST environment recognizes the distinction between general and specific parts of a software system. A design within the BOOST environment is regarded as being comprised of two parts:

1. An **information model** which represents the general aspects of a design.
2. A **behaviour model** which represents the application-specific parts of a design.

The information model is composed of a global view of the static representation of components of the software system (i.e. classes and class hierarchies), and is built during a stage which can be named *generic design*. The behaviour model is concerned with the dynamic relationships between objects, showing which objects are instantiated, how objects are composed and how they interact in the specific application. This model is created during what might be termed *specific design*.

This distinction between the generic and specific aspects of a design is an important feature which helps separate the object-oriented paradigm from other approaches to software production. The idea of being able to classify parts of a design as generic, and hence potentially reusable, is a powerful reason for maintaining this distinction and indeed for spending more time on the general aspects of the design than might really be needed for a specific application.

THE X MODEL

Software development is normally an iterative process, and the ability to easily navigate between different tools and notations is important to permit the software engineer to view concurrently different facets of a software system. The ability for a software engineer to navigate around is also vital as reusability is something that a CASE environment must promote. Software engineers must be able to browse through already-captured parts of previous software systems to try to see whether any component from prior work can be reused.

Presently, object-oriented reusable libraries with thousands of objects and class definitions are being used to build up all sorts of software systems. Nevertheless, one of the major problems which software engineers are faced with in trying to reuse software is the difficulty of finding reusable components, once such components have been produced. This is primarily because few mechanisms are available to help identify and relate components. In order to provide more convenient reuse, the question of which kinds of mechanisms might help solve this problem arises. The answer is typically couched in terms of finding components which provide specific functionality, from

0-7803-3019-6/96/$3.00 © 1996 IEEE.

Table 1: *BOOST Consortium Partners*

Institution Name	Country
MARI Computer Systems Ltd.	U.K.
IPSYS Software PLC	U.K.
University College of Wales	U.K.
Bull S.A.	France
Societe Francaise de Genie Logiciel	France
GIE Emeraude	France
Intrasoft S.A.	Greece
National Tech. Univ. of Athens	Greece
Intecs Sistemi	Italy
CWI	Netherlands
DeTeBerkom	Germany
SEL Alcatel	Germany
Universidade de Aveiro	Portugal
CET	Portugal
Telefonica	Spain

libraries of potentially reusable components linked through relationships which express their semantics, as described in [11, 12].

Traditional software life cycle models does not encourage reusability within their phases. Therefore, a software life cycle models which emphasizes the importance of reuse during software development is still needed. The X model as shown in Figure 1 has been proposed as a viable alternative.

Tools can manipulate reusable libraries by storing, selecting and browsing the potentially reusable components in these libraries. Selection involves browsing to find a component, retrieving it and transferring it from the reusable library to the developing software system, as earlier as while domain analysis is performed. On the other hand, before a component can be added to a reusable library, it must be validated and frozen. The validation is just applied to that particular component, not to the whole software system and should include treatment of exceptional conditions. Storing a component also involves classifying it first, getting it from the developing software system, relating it to other components and putting it into a reusable library.

The decisions involving the reuse of a component are very important in that the software engineer must select the component which requires the least effort to adapt, with an exact match between what is needed and what is available being the goal. Basically, the selection of a component from a reusable library involves four steps:

1. Identifying the required (target) component.
2. Selecting potentially reusable components.
3. Understanding the components.
4. Adapting (specializing, generalizing, composing or adjusting) the components to satisfy the needs of the developing software system.

The search for a component in a reusable library can lead to one of the following possible results:

- An identical match between the target and an available component is reached.
- Some closely matching components are collected, then adaptations are necessary.
- The design is changed in order to fit available components.
- No reusable component can be found, then the target component should be created from scratch.

While searching for components, it is necessary to address the similarity between the required (target) component and any near matching components. The best component selected for reuse may also require specialization, generalization or adjustment to the requirements of the new software system in which it will be reused. Sometimes, it is preferable to change the requirements in order to reuse the available components. The adaptability of the components depends

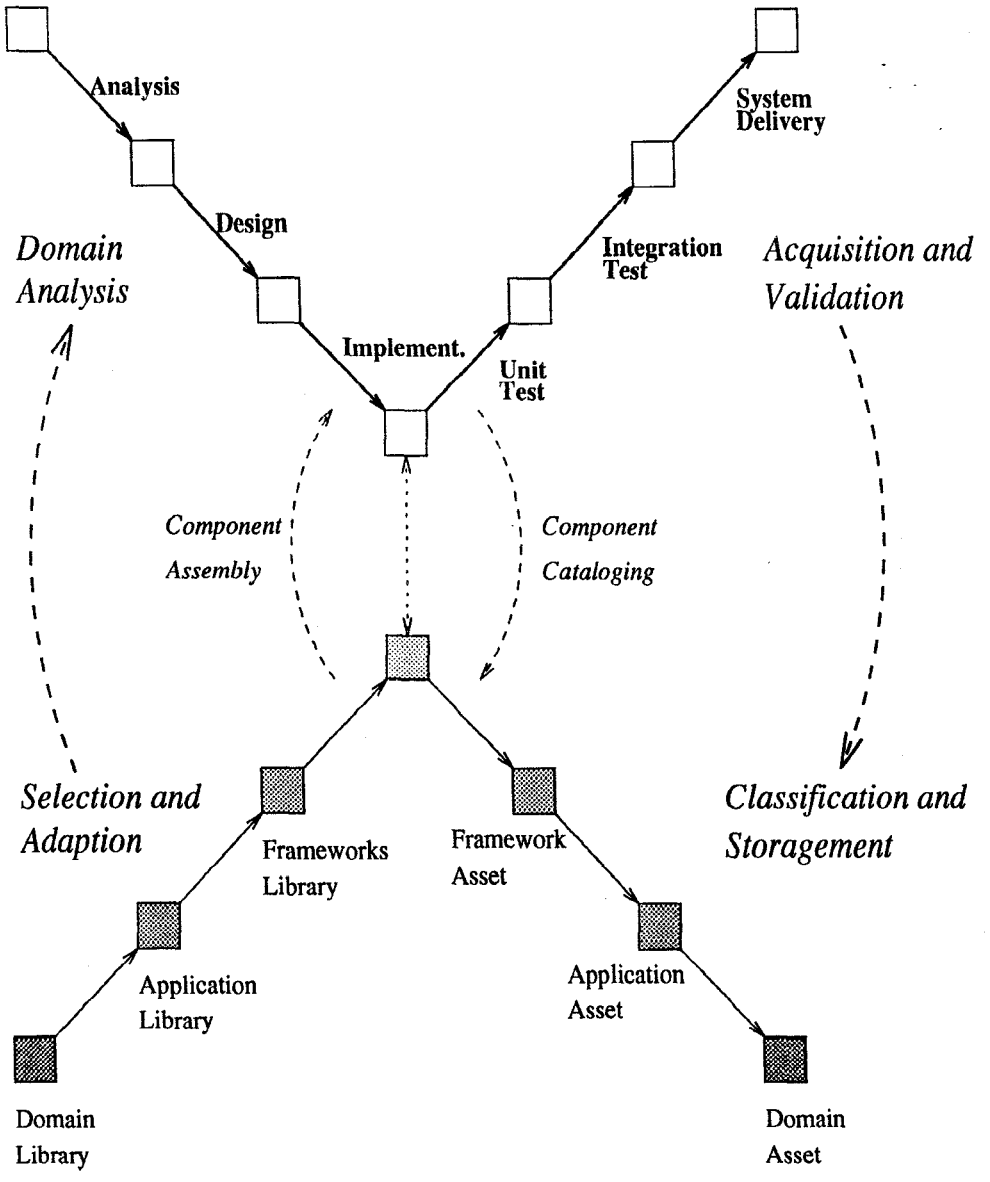

Figure 1: The X Model

on the difference between the requirements and the features offered by the existing components, as well as the skill and experience of the software engineer. The process of adapting components is the least likely to become automated in the software reuse process.

CONCLUSIONS

Software reusability is not very common in today's telecommunications industry. The BOOST project looked at the methodologies currently available for service creation and attempted to find a life cycle model which would be suitable for producing broadband services. This was difficult, bearing in mind that service engineering is still a new discipline. The X model has been used and evaluated by major European telecommunications service providers in the context of the BOOST project. The model has proved to cope with the inherent complexity of telecommunications services production.

So far, most browsing tools assume that component retrieval is a simple matter of matching well-formed queries to a reusable library. But forming queries can be demanding. A software engineer's understanding of the problem evolves while searching for a component, and large reusable libraries often use an esoteric vocabulary or jargon dependent on the application domain. Therefore, there is a need of new tools to support incremental query construction to yield a flexible retrieval mechanism that satisfies ill-defined queries and reduces the vocabulary problem.

Finally, based on available experience, the use of the X model appears to cover the likely phases of large software development and strongly supports software reuse. This is of paramount importance as reusability is believed to be a key factor to improve software development productivity and quality in the next few years.

References

[1] W. B. Frakes and S. Isoda, "Success Factors of Systematic Reuse," *IEEE Software*, vol. 11, no. 5, pp. 15–19, September 1994.

[2] L. F. Capretz and M. A. M. Capretz, **Object-Oriented Software: Design and Maintenance.** Singapore: World Scientific Publishing Co., ISBN 981-02-2731-0, 1996.

[3] G. Booch, **Object-Oriented Design with Applications.** Redwood City: Benjamin/Cummings, 1994.

[4] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, **Object-Oriented Modeling and Design.** Englewood Cliffs: Prentice Hall, 1991.

[5] R. M. Adler, "Emerging Standards for Component Software," *Computer*, vol. 28, no. 3, pp. 68–77, March 1995.

[6] D. D. Cowan and C. J. P. Lucena, "Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse," *IEEE Transactions on Software Engineering*, vol. 21, no. 3, pp. 229–243, March 1995.

[7] J. Markopoulos and et al., "Requirements on a Service Engineering Environment", *The BOOST Consortium, Deliverable to the CEC No R2076/MAR/MCS/DR/P/005/b0*, Brussels, 1993.

[8] P. Fitsilis, "Object-Oriented Development for Telecommunication Services," *Journal of Information and Software Technology*, vol. 37, no. 1, pp. 15–22, January 1995.

[9] S. Mazzini and et al., "Final Release of the BOOST Tools", *The BOOST Consortium, Deliverable to the CEC No R2076/MAR/MCS/DR/P/017/b0*, Brussels, October 1994.

[10] E. Sherrat and et al., "Assessment of BOOST Services with regard to the Target Environment", *The BOOST Consortium, Deliverable to the CEC No R2076/MAR/MCS/DR/P/016/b0*, Brussels, January 1995.

[11] L. F. Capretz and P. A. Lee, "Reusability and Life Cycle Issues within an Object-Oriented Methodology," in **TOOLS USA'92**, Englewood Cliffs: Prentice-Hall, pp. 139–150, 1992.

[12] L. F. Capretz and P. A. Lee, "Object-Oriented Design: Guidelines and Techniques," *Journal of Information and Software Technology*, vol. 35, no. 4, pp. 195–206, April 1993.