Measurement of Software Maintainability and Reusability in the Object Oriented Paradigm

Dr. Sallie Henry
Mark Lattanzi

Virginia Tech
Blacksburg, Virginia
September 6, 1994

1. Introduction

The Metrics Group of Virginia Tech has been studying the object oriented paradigm and how it relates to software maintainability and software reusability in an attempt to make programmers more productive. Software reuse is the key to increasing productivity within the software development process. By reusing existing software, time and effort are saved in the testing and maintenance phases of a software product. The object oriented paradigm is designed to enhance software reusability through encapsulation and inheritance [CoxB86, Meye87]. This paper describes the results of three studies on object oriented metrics and also a reusability study currently in progress. The first study investigates the maintainability of object oriented software versus procedural software. The second one examines the relationship between programmer productivity, software reuse, and the object oriented paradigm. The third study proposes and validates a new suite of object oriented metrics based on the MOOSE metrics developed by Chidamber and Kemerer [Chid91]. These new metrics are shown to predict maintainability from object oriented designs and source code.

A current study builds on the results of the research presented above. It involves the measurement of the reusability of object oriented software (classes) and how the process of reuse affects the software development lifecycle. In particular, can the implicit costs (retraining of programmers, learning object oriented design, searching libraries of reusable classes) and benefits (increased productivity, shortened testing time, easier maintenance) of software reuse be quantified?

2. Previous Work

The first study performed by the Virginia Tech Metrics group involving the object oriented paradigm concluded in 1988. Matthew Humphrey and Sallie Henry [Henr93] set up an empirical study with Computer Science seniors to determine if systems written in object oriented languages were more maintainable than procedural language based systems. A series of predefined maintenance tasks was constructed and given to the subjects. Humphrey showed that building applications with object oriented languages (like C++ or Objective C) results in final systems that are much more maintainable than systems constructed with procedural languages (like Pascal or C). This important result suggests the need for more research into the object oriented paradigm.

Study two validates another claim of the object oriented paradigm: that the object oriented paradigm promotes software reusability. John Lewis and Sallie Henry [Lewi92] show several significant results regarding the object oriented paradigm and software reusability. They conducted an experiment with Computer Science seniors to prove some of the benefits of software reuse. Lewis showed that the object oriented paradigm does, in fact, have an affinity towards reuse. Subjects using the object oriented languages reused more often and "better". Lewis also showed that software reuse improves programmer productivity regardless of the language used. However, the productivity increase was greater for subjects using object oriented languages. Lewis defined productivity as a function of development time and several other measurements relating to the cost of developing the software system. See [Lewi92] for his complete definition of productivity.

Wei Li headed the last study dealing with metrics in the object oriented paradigm. Li and Henry [LiWe93] developed some metrics specifically for object oriented languages that predict maintainability from code and, in some cases, from design documents. The research was validated using two commercial systems written in Classic Ada™ (object oriented Ada). The focus of the study was to bring together research in software metrics with research in the object oriented paradigm. Specifically, Li investigated a suite of proposed object oriented metrics [Chid91], proposed several new object oriented metrics, and validated the metrics using maintenance data collected from two commercial systems. See [Chid91] and [LiWe93] for descriptions of the ten object oriented metrics.

To validate their theories, data was collected over a three year period on the maintenance effort required by the two commercial systems. After performing the study, Li and Henry came to the following conclusions.

1. Maintenance effort can be predicted in the object oriented paradigm. The maintenance data gathered correlated highly with the some of the new object oriented metrics.
2. Maintenance effort can be predicted from metrics gathered from design documents. Although the new object oriented metrics are code metrics, some of the metrics can be generated at design time due to extensive nature of some of the common object oriented design documents (e.g. the class hierarchy chart).
3. There are observable patterns in the maintenance effort for object oriented systems. Since there are patterns in the maintenance efforts, predictions can be made as to how much maintenance a specific module of the system will require.
4. Object oriented metrics can be incorporated into an object oriented software development lifecycle. A seamless integration of the metrics gathering process and the software development process is essential if the new metrics are to be adopted by the software industry..

3. Current Research

The above three research efforts serve as a basis for a new research effort currently under way. Lewis investigated software reusability, but did not provide any quantitative results concerning when (under what conditions) to reuse or not. He also cites productivity gains, but not the quantitative costs and benefits of reusing software. This new research addresses these issues.

From Li's efforts, this new research leverages the proposed object oriented metrics and refines some of them into reusability metrics.

Software Reuse

Software reuse has been known about and studied for some time now [Bigg89, Free86, Trac90], but the software industry does not practice software reuse to its fullest potential. There are many reasons for this fact. The costs of reusing software are not well understood. Programmers must be retrained to design reusable software. Programmers must be trained to use reusable software effectively. Code written by many authors tends to be harder to read and understand. Other costs include the inefficiencies in execution speed brought about by reusing software and the fact that the software industry is profit based. Companies are reluctant to share code that they developed. Furthermore, there are always questions about the quality of code not written in-house.

The benefits of reusing software are also not well defined. Software reuse increases programmer productivity. How much? In what way? Are systems built from reusable components really easier to maintain? How much easier? Do testing requirements lessen for reusable modules? Other problems with reusing software include a lack of an industry standard for reusable modules. The object oriented paradigm helps solve this problem somewhat. Also, for reuse to be effective, large class libraries are required. What classes should be incorporated into these libraries? How should these libraries be organized? What is a good size for a reusable class? What methods and data should it contain? For a more detailed explanation on why software reuse hasn't been incorporated into the software community, see [Bigg87, Free86, Meye87, Trac87, Trac88b].

Description of New Research

This new research, headed by Mark Lattanzi, attempts to quantify some of the properties of reusable software in hopes that some of the reluctance to reuse software can be clarified and dispelled.. Object oriented software (classes) is by nature reusable, but there are few, if any, metrics that attempt to measure the reuse potential of one of these classes in these libraries (or even the reuse potential of a procedural language library). This new research attempts to characterize reusability and subdivide it into measurable properties. Part of this research is to quantitatively measure such concepts as productivity and reusability. Also, it investigates what aspects of software promote reuse and how software reusability affects the software development process and the programmers involved in it. Korson [Kors92] authored a paper on the characteristics of reusability. Some of the metrics to be developed are based on these traits. The rest of the new metrics focus on programmers' opinions regarding the classes in the libraries and their reuse potential.

Specifically, the goals of this ongoing research are:

GOAL 1: to quantitatively measure object oriented designs and relate them to aspects of the final code (complexity, reusability, readability),

GOAL 2: to identify the measurable qualities of reusable software (costs, benefits),

GOAL 3: to correlate various aspects of reusability with programmer characteristics,

GOAL 4: to measure the effects of reusing software on the process of software development (design, coding, integration, testing, maintenance).

The first goal relates to the object oriented paradigm and the role it plays in reusing software. Although reuse is the key issue of this research; it is beneficial to be able to measure design documents and use the measures to predict many traits of the final product. Therefore, this goal's focus is on prediction at design time of the attributes of the final product (code) including complexity, maintainability, and reuse potential of the various components of the final code. This goal is based on the work done by Li and Henry [LiWe93].

The second goal deals with the characteristics of reusable components and systems built from reusable components. These characteristics include code complexity, code readability, and specific attributes of reusable components that make them easy to reuse (number of parameters, level of abstraction, number of methods, etc.). Under this goal, this research attempts to quantify the costs and benefits to reuse code in several different situations such as black box reuse, white box reuse, and no reuse. Under white box reuse, the goal is to quantify when the reuse benefit is lost, i.e. how much code modifying is too much?

Goal three relates to the human factors side of software reuse. Tracz [Trac88a, Trac88b] and others [Free86, Hoop91, Lewi92, Wood87] point out that software reuse is not just a technical problem. People are reluctant to reuse. Some programmers are reluctant to reuse other people's code. Some programmers are good at writing reusable code; others are good at reusing existing code. To meet this goal, data on the programmers involved in the experiment is gathered and data on the code that each programmer writes or reuses is recorded. Statistical correlations can then be examined.

The last major goal of this research is to measure the effect that reusing software has on the stages of the software development lifecycle. Lewis [Lewi92] did some preliminary work in this area proving several important results concerning software reuse.

In summary, this new research attempts to further quantify the problems involved in developing reusable software in the object oriented paradigm and to provide some tools with which to combat these problems.

6. Conclusions

The three previous research efforts and the ongoing one all deal with the object oriented paradigm and its relationship to the software development process. The object oriented paradigm is claimed to aid in software reuse and software maintainability. These research efforts support these claims. But, there is still far too little evidence for the software industry to adopt using the object oriented paradigm. Furthermore, there is a lack of object oriented software metrics which are essential for controlling software quality and cost. The Virginia Tech Metrics group is involved in ongoing research efforts to remedy this situation.

7. References

[Bigg87] Biggerstaff, Ted and Charles Richter, "Reusability Framework, Assessment, and Directions," *IEEE Software*, March 1987.

[Chid91] Chidamber, S.R. and Chris Kemerer, "Towards a Metrics Suite for Object Oriented Design," *OOPSLA 1991*, pp. 197-211.

[CoxB86] Cox, Brad and Andrew Novobiliski, Object Oriented Programming: An Evolutionary Approach, Addison-Wesley, Reading, MA 1986.

[Free86] Freeman, Peter, Tutorial:, Computer Society Press of the IEEE, Washington D.C., 1986.

[Henr93] Henry, Sallie and Matthew Humphrey, "Comparison of an Object Oriented Programming Language to a Procedural Programming Language for Effectiveness in Program Maintenance," *Journal of Object Oriented Programming*, to appear 1993.

[Kors92] Korson, Tim and John McGregor, "Technical Criteria for the Specification and Evaluation of Object Oriented Libraries," *Software Engineering Journal*, March 1992.

[Lewi92] Lewis, John, Sallie Henry, Dennis Kafura, and Robert Schulman, "On the Relationship Between the Object Oriented paradigm and Software Reuse: An Empirical Investigation," *Journal of Object Oriented Programming*, August 1992.

[LiWe93] Li, Wei and Sallie Henry, "Object Oriented Metrics Which Predict Maintainability," To appear in a special issue on the object oriented paradigm, *Journal of Systems and Software*, November 1993.

[Meye87] Meyer, Bertrand, "Reusability: The Case for Object Oriented Design," *IEEE Software*, March 1987, pp 50-64.

[Trac87] Tracz, Will, "Reusability Comes of Age," *IEEE Software*, July 1987.

[Trac88a] Tracz, Will, "Software Reuse Maxims," *ACM SigSoft Notes, Software Engineering Notes*, 13(4), 1988, pp. 28-31.

[Trac88b] Tracz, Will, "Software Reuse Myths," *ACM SigSoft Notes, Software Engineering Notes*, January 1988.

[Trac90] Tracz, Will, Tutorial: Software Reuse: Emerging Technology, The Computer Society Press of the IEEE, Washington D.C., 1988.

[Wood87] Woodfield, Scott, David Embley, Del Scott, "Can Programmers Reuse Software?" *IEEE Software*, July 1987.