Reuse Metrics for Object Oriented Software

Summary of Research
Final Report

Supported by NASA Langley Research Center
Grant number NAG1-1461
Fiscal Years 1993–1997
(December 1, 1992 – November 1, 1997)

Principal Investigator
James M. Bieman, Associate Professor
Department of Computer Science
Colorado State University
Fort Collins, CO 80523
(970) 491-7096 bieman@cs.colostate.edu

January 23, 1998

Abstract

One way to increase the quality of software products and the productivity of software development is to reuse existing software components when building new software systems. In order to monitor improvements in reuse, the level of reuse must be measured. In this NASA supported project we (1) derived a suite of metrics which quantify reuse attributes for object oriented, object based, and procedural software, (2) designed prototype tools to take these measurements in Ada, C++, Java, and C software, (3) evaluated the reuse in available software, (4) analyzed the relationship between coupling, cohesion, inheritance, and reuse, (5) collected object oriented software systems for our empirical analyses, and (6) developed quantitative criteria and methods for restructuring software to improve reusability.

Prior to this project, we defined classes and perspectives of software reuse, proposed relevant reuse abstractions, attributes and metrics applicable to object oriented systems [20]. During this NASA supported research project, we defined a suite of reuse metrics and developed prototype measurement tools, derived measures of cohesion and inheritance structure, began the development of a benchmark set of software for empirical study, measured reuse in the benchmark software to determine the way inheritance is actually used and examined the conflict between reuse and coupling. Our results are strongly tied to fundamental issues in software measurement [1, 3]. We have also studied the use of formal specifications in the retrieval of reusable software components, and the use of reusable assertions in software testing. See Section 9.1 on page 4 for a list of our publications describing results from this project.

1 Measurement Tools

We have developed measurement tools for Ada, C++, Java, and C software.

Ada Reuse Measurement Analyzer (ARMA). ARMA takes measurements specified in terms of Ada constructs such as overloading, generics and derived types. Measures include the number and size of direct and indirect servers and clients, server instance creations, server methods, server method instances, and direct client invocations of servers [6, 14]. ARMA makes use of the Anna Tool set developed at Stanford [27]. ARMA was tested on a software library supplied by CTA, Inc.

C++ Inheritance Analyzer (Jasmin). Jasmin analyzes inheritance and reuse in C++ class libraries [8]. It measures the size and shape of the inheritance hierarchy graph of a C++ system, and the inheritance characteristics of classes with and without private sections, and with private or public inheritance.

Gen++-Based C++ Tools (Inheritance Structure and Class Cohesion). We developed a set of measurement tools using Gen++ from AT&T, which makes use of the GENOA language [24]. Using Gen++ we can quickly implement complex measurement tools. We developed one tool that duplicates the inheritance structure capabilities of Jasmin and another that measures the cohesion of C++ classes [4].

Celebes Java Class Cohesion Tool. We applied our class cohesion measures [4] to Java programs [18]. *Celebes* performs static analysis on source code and Java byte codes. *Celebes* can be extended to implement other forms of static analysis and instrumentation.

Functional Cohesion Measurement Tool (Funco) We developed the prototype Funco tool which is designed to measure functional and design-level cohesion in C programs. Funco implements a set of three functional cohesion measures [7] and four design-level cohesion measures [10, 5]. Our design-level cohesion measures can be used to re-structure a design to improve cohesion [9, 11, 12] and to predict the functional cohesion in an implementation [5].

Funco can measure *Strong functional cohesion (SFC)*, which is closely related to Yourdon and Constantine's original notion of functional cohesion [29, ch. 7], *Weak functional cohesion (WFC)*, and *Adhesiveness*. We appear to be the first to publish rigorously defined, ordinal scale functional cohesion measures [7].

The design-level measures are based on an analysis of interface information [5, 10]. One set of measures is adapted from our functional cohesion measures. The other measure is derived from Lakhotia's cohesion measure [26].

2 Benchmark Software

We collected sample software to evaluate our tools and examine actual reuse. Some of the software is written in Ada; most of it is written in C++. We collected 19 C++ public domain systems with more than 2,700 classes and 265,000 non-commented source lines of source code. Some of the systems were designed for reuse—the classes are to be used when developing other software, while others are applications designed for customer use. Included are some commonly referenced reuse libraries such as `InterViews`, `Motif_C++`, and

the National Institute for Health Class Library (NIHCL). The systems include programming language tools, graphical user interfaces (GUI) software, threads and parallel programming packages, and other applications.

We use our collected object oriented software data base as a benchmark for future measurement and analysis work. The test bed software can serve as a generally available benchmark for research on object oriented software measurement. Other researchers have used our collected software [22, 23].

3 Analyzing the Benchmark Software

Actual Use of Inheritance. Inheritance is an excellent way to organize abstraction and support reuse, but it has costs—inheritance increases the complexity of a system and the coupling between classes.

We used Jasmin to measure the class depth in inheritance hierarchies, and the number of child and parent classes in the benchmark software [8]. Inheritance is used extensively in many of the systems, while it is used infrequently in others. The average depth of inheritance for classes is less than 1.0 in 60% of the systems. The greatest use of inheritance is in the GUI applications, perhaps because GUI applications tend to model a hierarchical world of user interface objects such as icons and windows. Reuse library classes use inheritance much more than the applications system classes. The measurements also indicate that there are many classes that exist independently with no parent or child classes. Few of the systems have the 7 +- 2 maximum class inheritance depth recommended by Booch [21].

Class Cohesion and Reuse. We explored alternatives for deriving measures of class cohesion [16]. We developed two class cohesion measures based on the connectivity of class methods [4]. Methods can be connected directly through shared instance variables, or indirectly through a chain of directly connected methods. *Tight class cohesion* (TCC) is the relative number of directly connected methods in a class. *Loose class cohesion* (LCC) is the relative number of indirectly and directly connected methods.

We examined, in depth, the properties of our class cohesion measures [19]. These measures generally do satisfy the representation condition of measurement — they are consistent with properties defined by their empirical relation system. However, they do not completely reflect the ability to split a class without breaking method connections. We propose a new approach to measuring factorability, and demonstrate the difficulties of deriving connectivity-based cohesion measures that completely satisfy the representation condition with respect to factorability.

We applied the cohesion measures to the InterViews system, which consists of more than 25,000 non-commented source lines of C++, and analyzed the relationship between cohesion and private reuse [4]. We found no relationship between cohesion and reuse through instantiation. However, we found a statistically significant (and counter-intuitive) relationship between cohesion and reuse through inheritance. Classes with lower cohesion values tend to have more descendants.

The Conflict Between Reusability and Coupling. When designing object oriented software, developers must deal with a conflict between the advantages of inheritance (increased reuse, and improved similarity of implementation and problem structure) and disadvantages (increased coupling and complexity). Because of this conflict, developers should limit inheritance and reduce coupling. We find that developers do limit the use of inheritance.

Rising and Calliss developed an ordinal set of categories of coupling in Ada software [28]. We extended these definitions to develop the notion of class coupling, and show both analytically and empirically that the conflict between reuse and coupling is real [17]. Our data suggests that two design practices, (1) dividing a system into general-purpose and specialized modules, and (2) using multiple inheritance in appropriate situations, can result in greater reuse with minimal increases in coupling.

4 Restructuring to Improve Reusability.

Existing software systems may benefit from restructuring to improve maintainability and reusability. Yet, intuition-based, *ad hoc* restructuring can be difficult and expensive, and may make software structure worse.

We introduced a quantitative framework for software restructuring which provides objective criteria for restructuring decisions [9, 11, 12]. The framework makes use of a formal representation of design structure

that can be readily displayed visually. Engineers can extract design information directly from code to restructure existing or legacy software. Criteria for restructuring include measures of design-level cohesion and coupling; other quantitative criteria can be included. Restructuring is accomplished through a series of decomposition and composition operations which increase the cohesion and/or decrease the coupling of individual system components. Engineers make restructuring decisions using both quantitative information and a visual display of the design structure. The process insures that restructuring results in measurable and visible improvements in design quality.

5 Inheritance Tree Shapes and Reuse

The shapes of forests of inheritance trees can affect the amount of code reuse in an object-oriented system [13]. We show that a set of objective measures can classify forests of inheritance trees into a set of five shape classes. These shape classes determine bounds on reuse measures based on the notion of code savings. The reuse measures impart an ordering on the shape classes that demonstrates that some shapes have more capacity to support reuse through inheritance. An initial empirical study shows that the application of the measures and demonstrates that real inheritance forests can be objectively and automatically classified into one of the five shape classes.

6 Component Retrieval for Reuse

We developed a pragmatic technique to use formal specifications as keys when searching for software components in a reuse data base [15]. The technique uses a relaxed specification match—first order predicate calculus specifications are mapped to propositional calculus formula before attempting a match—and a partial ordering of specifications. A prototype system demonstrates the effectiveness of the method. We use the expressive power of first order predicate calculus, but avoid undecidability problems by introducing approximate retrieval. Initial tests indicate that only rarely will more than one component will be retrieved by any single request. Users will not have to exmine many similar candidates for reuse suitability.

7 Reusable Assertions for Testing

Our prototype system, Visual C-Patrol (VCP), supports the application of reusable assertions during software testing [2]. Assertions are written as *virtual code* which is inserted at specified locations in a C program. These assertions can act as test oracles that determine the correctness of program execution. The system is designed to maximize the potential to reuse assertions by using parameterized assertions. VCP also uses *fault injection* to force program execution to follow particular paths during testing.

8 Summary of Technology Transfer

Funco is being used as a research tool at Colorado State and other locations including Michigan Technological University. The Funco software and documentation can now be accessed over the world wide web; the Funco home page, `http://www.cs.colostate.edu:80/~bieman/funco.html`, has been accessed more than 1100 times since it was created in 1996 and the downloading instructions have been accessed by more than 300 potential users during the same period.

We plan to add demonstration web access to allow Funco to be tried without the need to download and install the system on clients. We also plan to make the prototype Celebes system available over the web.

Our benchmark software has been used by researchers working on improved object-oriented compiler design [22, 23].

9 References

9.1 Publications from Our NASA Supported Research

The following are publications from research that was supported or partially supported by NASA Langley Reseearch Center grant number NAG 1 1461.

[1] J. Bieman. Metric development for object-oriented software. In *Software Measurement: Understanding Software Engineering.* A. Melton, editor, Int. Thompson Computer Press, pp. 75–92, 1996.

[2] J. Bieman, D. Dreilinger and L. Lin. Using fault injection to increase software test coverage. *Proc. Int. Symp. Software Reliability Engineering (ISSRE'96)*, Oct. 1996.

[3] J. Bieman, N. Fenton, D. Gustafson, A. Melton, and L. Ott. Fundamental issues in software measurement. In *Software Measurement: Understanding Software Engineering.* A. Melton, editor, Int. Thompson Computer Press, pp. 39–74, 1996.

[4] J. Bieman and B-K. Kang. Cohesion and reuse in an object-oriented system. *Proc. ACM Software Reusability Symp. (SSR'94)*, pp. 259–262, April 1995. Reprinted in *ACM Software Engineering Notes*, Aug. 1995. Runner-up for best concise paper in SSR'94.

[5] J. Bieman and B-K. Kang. Measuring design-level cohesion. *IEEE Trans. Software Engineering*, To Appear.

[6] J. Bieman and S. Karunanithi. Measurement of language supported reuse in object oriented and object based software. *Journal of Systems and Software*, 28(9):271–293, Sept. 1995.

[7] J. Bieman and L. Ott. Measuring functional cohesion. *IEEE Trans. Software Engineering*, 20(8):644–657, Aug. 1994.

[8] J. Bieman and J.X. Zhao. Reuse through inheritance: A quantitative study of c++ software. *Proc. ACM Software Reusability Symp. (SRS'94)*, pp. 47–52, April 1995. Reprinted in *ACM Software Engineering Notes*, Aug. 1995.

[9] B-K. Kang. A Quantitative Framework for Software Restructuring. Ph.D thesis, Department of Computer Science, Colorado State University, 1997.

[10] B-K. Kang and J. Bieman. Design-level cohesion measures: derivation, comparison, and applications. *Proc. 20th Int. Computer Software & Applications Conf (COMPSAC'96)*, pp. 92-97, Aug. 1996.

[11] B-K. Kang and J. Bieman. Using design cohesion to visualize, quantify, and restructure software. *Proc. 8th Int. Conf. Software Engineering and Knowledge Engineering (SEKE'96)*, pp. 222-229, June 1996.

[12] B-K. Kang and J. Bieman. Using design abstractions to visualize, quantify, and restructure software. *J. Systems and Software*, To Appear.

[13] B-K. Kang and J.Bieman. Inheritance tree shapes and reuse. *Proc. 4th Int. Software Metrics Symp. (Metrics'97)*, pp. 34–42, Nov. 1997.

[14] S. Karunanithi and J. Bieman. Candidate reuse metrics for object oriented and Ada software. *IEEE-CS Int. Symp. Software Metrics*, 1993.

[15] L. Lin and J. Bieman. Retrieving functions for software reuse: A pragmatic approach to specification matching. submitted to *Proc. Fourth Int. Conf. Software Reuse (ICSR-4)*, April 1996.

[16] L. Ott, J. Bieman, B-K. Kang, and B. Mehra. Developing measures of class cohesion for object-oriented software. *Proc. Annual Oregon Workshop on Software Metrics (AOWSM'95)*, June 1995.

[17] L. Wu. Coupling and reuse in object oriented systems. Master's thesis, Department of Computer Science, Colorado State University, 1994.

[18] M. Shumway. Measuring class cohesion in java. Master's thesis, Department of Computer Science, Colorado State University, 1997. Available as Computer Science Technical Report CS-87-113 URL `http://www.cs.colostate.edu/~ftppub/TechReports/1997/tr97-113.ps.Z`.

[19] M. Shumway, J. Bieman, and S. Seidman. Validating the Bieman-Kang class cohesion measures. *IEEE Trans. Software Engineering*, Submitted for publication.

9.2 Other References

[20] J. Bieman. Deriving measures of software reuse in object-oriented systems. In T. Denvir, R. Herman, and R. Whitty, editors, *Formal Aspects of Measurement. (Proc. BCS-FACS Workshop on Formal Aspects of Measurement)*, pp. 79–82. Springer-Verlag, 1992.

[21] G. Booch. *Object-oriented Analysis and Design with Applications 2nd Edition.* Benjamin/Cummings, Redwood City, CA, 1994.

[22] C. Chambers, J. Dean, and D. Grove. *Whole-Program Optimization of Object-Oriented Languages.* Computer Science Technical Report 96-06-02, University of Washington, June 1996.

[23] J. Dean, G. DeFouw, D. Grove, V. Litvinov, C. Chambers. "Vortex: An Optimizing Compiler for Object-Oriented Languages." *Proc. OOPSLA'96*, San Jose, CA, October, 1996.

[24] P. Devanbu. GENOA a customizable, language- and front-end independent code analyzer. *Proc. Int. Conf. Software Engineering (ICSE)*, pp. 307–317, 1992.

[25] N. Fenton. *Software Metrics A Rigorous Approach.* Chapman & Hall, London, 1991.

[26] A. Lakhotia. Rule-based approach to computing module cohesion. *Proc. 15th Int. Conf. Software Engineering*, pages 35–44, 1993.

[27] D. Luckham, editor. *Programming with Specifications: An Introduction to Anna, A language for specifying Ada programs.* Springer-Verlag, 1990.

[28] L. Rising and F. Calliss. Problems with determining package cohesion and coupling. *Software Practice and Experience*, 22(7):553–571, July 1992.

[29] E. Yourdon and L. Constantine. *Structured Design.* Prentice-Hall, Englewood Cliffs, NJ, 1979.