

Software Design Quality: Style and Substance

Position Paper

James M. Bieman

Roger Alexander

P. Willard Munger III

Erin Meunier

Computer Science Department

Colorado State University

Fort Collins, Colorado 80523

bieman@cs.colostate.edu

ABSTRACT

Many software development texts, references, tools, and authorities provide advice on good software design and programming styles. Unfortunately, most of the evidence to support the value of this advice consists of intuition and anecdotes. We are working to objectively determine the value of recommended style guides on large-scale real world software systems; we are studying both proprietary commercial and open source systems. Our work involves determining whether or not style recommendations are followed, and how these styles affect external quality factors such as fault- and change-proneness, and maintainability. Early results indicate that style guidelines are often violated. In addition, we have found that, in contrast with common claims, one design recommendation — the use of design patterns — can lead to more change prone, rather than less change prone classes.

1. INTRODUCTION

From early in the history of computer software, a variety of sources have provided guidance on how to design and code to improve program understandability and adaptability and decrease faults. Dijkstra's "Go To Statement Considered Harmful" letter to CACM" is probably the most cited early programming style advice [4]. Marshall and Webber point out other common "programmer's taboos" including avoiding (1) low level programming, (2) the use of flowcharts, (3) pointers, and (4) platform specific programming [7].

Additional style advice is specific to the object-oriented (OO) paradigm. For example, OO developers are urged to apply design patterns [6]. Fowler, in his popular refactoring book, recommends that programmers look for 22 "bad smells" in their code, and refactor the code to remove them [5].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WoSQ - Workshop on Software Quality 2001 Orlando, USA
Copyright 2002 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

A number of available tools can find style violations. Jstyle, from Man Machine Systems, can identify at least 45 distinct style "errors" in Java programs. Together, from TogetherSoft, can identify at least 17 "high severity", 19 "normal severity", 29 "low severity" style violations. You can find a long lists of style recommendations by searching *google.com* using the keywords "java programming style".

Marshall and Webber [7] are concerned that such style "taboos" can "needlessly restrict programming options. Actually the benefits, and costs, of following recommended styles are unknown.

In addition to the many guidelines concerning good program style, there is a commonly-held belief that most software is of low quality [3].

Our work focuses on assessing the quality of software designs, as well as developed software. Ultimately, we are concerned with external quality factors such as reusability, maintainability, testability, and adaptability. These are the factors that are visible to either or both software users and developers. We want to determine the relationship between design structures and external quality factors such as reusability, maintainability, testability, and adaptability.

The underlying objective of our ongoing research is to answer, in an objective manner, two key questions:

1. What are good ways to structure software?
2. How is real-world software structured, and what can we say about its "structural quality"?

We look for answers, in large part, by examining real software systems — systems that people actually use and have evolved over a significant period of time.

2. DESIGN STRUCTURE AND QUALITY

We have been studying the relationship between design structure and external quality of proprietary as well as open source software systems. Some of the work is descriptive in nature; other work focuses on identifying the relationships between structure and external qualities.

In one case study [1], we analyze 39 versions of an evolving industrial OO software system to see if there is a relationship between patterns, other design attributes, and the number of changes. We found a strong relationship between

class size and the number of changes — larger classes were changed more frequently. We also found two relationships that we did not expect: (1) classes that participate in design patterns are **not** less change prone — these pattern classes are among the most change prone in the system, and (2) classes that are reused the most through inheritance tend to be more change prone. These unexpected results hold up after accounting for class size, which had the strongest relationship with changes. In follow-up work, we developed a *change architecture* which models the relationships among the most change prone system components.

This work is continuing with a study of a much larger industrial OO system — a multi-version system with more than 700 classes. We will see if the results from the first study [1] are repeated with the new data.

We have applied the Jstyle and Together style analyzer tools to both proprietary and open source code. We have found large numbers of style “violations” in this code. The impact of these violations on external quality is an open question, which will require further study. Our aim here is to demonstrate the value or lack of value of the style guidelines and structuring rules.

3. SOFTWARE ENGINEERING ARTIFACTS AS DATA

Studies of software structure, style, and quality depend upon having real-world software development artifacts available for study. This software engineering data can include program code, designs, specifications, models, etc. In the past, such data has been very difficult to obtain. Proprietary concerns prevent many organizations from giving researchers access.

We have had some success in obtaining proprietary software artifacts. However, with the growth of the web and open source software, software artifacts (at least code) are widely available. The Web contains at least 3.4 million files containing either Java, C++, or Perl source code, and at least 8.7 million files contain executable code [2]. Much of this software can be classified as *open source*, which means that it is freely available for both use and research. Open source software often includes multiple versions which are managed by the Concurrent Version System (CVS). CVS supports the analysis of transitions between software versions.

The following lists some commonly available large sized systems, with the storage required for their CVS systems:

- NetBeans: ~ 1.2 GByte.
- Jakarta: ~ 5.4 GByte.
- Linux Kernel: ~ 14 GByte.
- KDE: ~ 5.4 GByte.

The foregoing software artifacts are all program code with change information and trouble reports that are logged on the CVS system.

4. IS OPEN SOURCE SOFTWARE HIGH QUALITY?

A common belief in the software development community is that open source software is better than proprietary code.

We intend to examine this question. Our investigation will help to determine whether or not open source software is error prone, easy to understand and maintain.

One key concern is how open source software ages. As software evolves, the entropy tends to increase. Dependencies between components can increase, making it more difficult to make changes, and allowing errors to propagate.

There is some evidence that coupling in open source code can increase dramatically. Schach et al studied the evolution of the open source Linux kernel through 365 versions [8]. They found that common coupling between kernel modules is growing at an exponential rate, while the kernel module size is growing linearly. Eventually the kernel may become unmaintainable, requiring a major redesign effort.

We plan to examine the Linux kernel and other open source systems to further examine whether coupling, and other negative properties, are increasing at an alarming rate.

5. CONCLUSIONS

Advise on good program design structure and good programming practice should be based on objective studies. Such studies depend upon large sets of data consisting of software artifacts. These software artifacts should be generally available so that the research community can use the same benchmarks to evaluate new tools, techniques, and advise. We are working towards developing a software engineering research repository by collecting and cataloging software artifacts. The repository will be available as a resource for the software engineering research and practitioner community.

Acknowledgements

This work is partially supported by U.S. National Science Foundation grant CCR-0098202, and by a grant from the Colorado Advanced Software Institute (CASI). CASI is sponsored in part by the Colorado Commission on Higher Education (CCHE), an agency of the State of Colorado.

Author Information

Jim Bieman is the Editor-in-Chief of the *Software Quality Journal* and Associate Professor of Computer Science at Colorado State University. His work is focused on the evaluation and improvement of software design quality.

Roger Alexander is an Associate Professor of Computer Science at Colorado State University. He spent many years in industry as a software developer (and researcher) including experience at the Software Productivity Consortium, Michael Jackson Systems, and Cigital (formerly Reliable Software Technologies).

Willard Munger has taught Computer Science at in Rwanda and Camaroon; he was the founding President of Adventist University Cosendai in Camaroon. He is now on sabbatical at Colorado State University.

Erin Meunier is an undergraduate Research Assistant at Colorado State University. She will soon receive her B.S. in Computer Science and will be joining IBM.

6. REFERENCES

- [1] J. Bieman, D. Jain, and H. Yang. Design patterns, design structure, and program changes: an industrial case study. *Proc. Int. Conf. on Software Maintenance (ICSM 2001)*, 2001.

- [2] J. Bieman and V. Murdock. Finding code on the world wide web: a preliminary investigation. *Proc. Int. Workshop on Source Code Analysis and Manipulation (SCAM 2001)*, November 2001.
- [3] C. Connell. Most software stinks. URL <http://www.chr-3.com/pub/beautifulsoftware.htm>, 2001.
- [4] E. Dijkstra. Go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, 1968.
- [5] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Reading MA, 1999.
- [6] E. Gamma, R. Helm, J. R., and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading MA, 1995.
- [7] L. Marshall and J. Webber. Gotos considered harmful and other programmers' taboos. *Proc. 12th Workshop Psychology of Programming Interest Group (PPIG-12)*, 2000.
- [8] S. Schach, B. Jin, D. Wright, G. Heller, and J. Offutt. Maintainability of the linux kernel. *IEE Proceedings Journal: Special Issue on Open Source Software Engineering*, 2002.