Why do Developers not Take Advantage of the Progress in Modularity?

Poster

Paul Leger
Universidad Católica del Norte, Chile
pleger@ucn.cl

Hiroaki Fukuda
Shibaura Institute Technology, Japan
hiroaki@shibaura-t.ac.jp

ABSTRACT

Modular software development has several benefits, such as flexibility regarding unexpected changes and increasing our ability to understand each module in isolation. Over the last few years, research on modular development has resulted in many proposals, from new abstractions to the creation of new programming paradigms. However, developers of real projects have not implemented these proposals in their daily work. In fact, in most cases, developers keep using their basic knowledge of object-oriented programming. In this poster, we discuss some possible reasons for this problem.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features; D. 2.10 [**Software Engineering**]: Design.

General Terms

Languages, Design.

Keywords

Programming languages, Design, modularity.

1. INTRODUCTION

To create software, developers must implement the set of *concerns* that composes the software. Developing these concerns in a maintainable and evolvable manner is crucial for the software survival the software over time. To achieve this goal, *modular programming* [4,7] has proven useful because it allows developers to:

- Make the inclusion of new implementations of a concern more flexible. For example, a system's security policy update does not require modifying the implementation of the reporting concern.
- Implement concerns in a parallel manner. For instance, it is possible to implement security policy and reporting concerns simultaneously.
- Understand concern implementations in an isolated way. For example, to understand the implementation of the reporting concern, developers do not have to understand the implementation of the security policy concern.

Modular programming concretely means that a program should be decomposed in a set of *modules* and that each module should address a given concern of the software [4]. To increase modularity in software development, a number of techniques have been proposed. These techniques range from new abstractions such as Mixin [3], Traits [8], and Classboxes [2] to new paradigms such as Aspect-Oriented Programming (AOP) [6].

2. PROBLEM STATEMENT AND DISCUSION

Despite the evident progress in the modularity of software development, developers keep using basic concepts of modularity. For example, regarding object-oriented programming, some studies [9, 10] show that developers make design flaws (e.g., "antipatterns"). Another example is the adoption of AOP in the software industry. Although AOP was proposed 18 years ago (1996) to modularly implement *crosscutting concerns* [6] and there are practical aspect languages (*e.g.,* AspectJ [1]), developers continue wondering if this paradigm is useful for real applications [11]. Considering this evidence, we wonder *why developers do not take advantage of the progress in modularity.*

This question might have different answers; however, we think the following variables are important:

1. Time. In the current and competitive market, a computing system must be implemented as soon as possible. Thus, developers do not have sufficient time to think through their work.

2. Knowledge. Applying new modularity techniques requires knowledge that is not part of most computer science curricula.

3. Consequences. Recent modularity techniques are still being studied, implying that some consequences are yet unknown. Therefore, developers still feel afraid to apply these techniques in real projects.

4. Need. Although researchers in this area give reasons to modularly implement systems, developers think these reasons are not strong enough to support full modularity in their systems. For

BICT 2014, December 01-03, Boston, United States

DOI 10.4108/icst.bict.2014.257938

example, the *observer pattern* [5], which mixes subject and observer concerns, is widely adopted and used.

5. Working environment. A developer has commonly to follow organization rules, which are defined previously. If these rules do not permit innovating with new modularity techniques, developers cannot apply these techniques.

To the best of our knowledge, few studies try answering this question, maybe because of complexity of getting data for development industry. In practice, it is possible to find some frameworks that internally use advanced modularity techniques (*e.g.,* AOP in Spring); however, these libraries hide this fact from developers through simple interfaces.

3. PLAN

We propose to answer the paper's question with a study that is focused on the five variables mentioned above. To carry out this study, we plan conduct a survey to a wide set of developers. With the survey's result, we will use models of human behavior like Technology Acceptance Model (TAM) [12] to relate the five variables.

REFERENCES

[1] Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Jennifer Lhoták, Ondrej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. abc: an extensible AspectJ compiler. In *Transactions on Aspect-Oriented Software Development*, volume 3880 of Lecture Notes in Computer Science, pages 293-334. Springer-Verlag, 2006.

[2] Alexandre Bergel. Classboxes - controlling visibility of class extensions. *it-Information Technology*, 49(4), July 2007.

[3] Gilad Bracha and William Cook. Mixin-based inheritance. In Norman Meyrowitz, editor, *Proceedings of the 5th International Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA/ECOOP 90)*, pages 303-311, Ottawa, Canada, October 1990. ACM Press. ACM SIGPLAN Notices, 25(10).

[4] Edsger W. Dijkstra. The structure of the THE multiprogramming system. *Communications of the ACM*, 11(5): 341–346, May 1968.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, October 1994.

[6] G. Kiczales, J. Irwin, J. Lamping, J. Loingtier, C.V. Lopes, C. Maeda, and A. Mendhekar. Aspect oriented programming. In Special Issues in *Object-Oriented Programming*. Max Muehlhaeuser (general editor) et al., 1996.

[7] David Parnas. On the criteria for decomposing systems into modules. *Communications of the ACM*, 15(12): 1053-1058, December 1972.

[8] Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz, and Andrew Black. Traits: Composable units of behavior. In Luca Cardelli, editor, *Proceedings of the 17th European Conference on Object-Oriented Programming (ECOOP 2003),* number 2743 in Lecture Notes in Computer Science, pages 248–274, Darmstadt, Germany, July 2003. Springer-Verlag.

[9] Radu Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *Software Maintenance 2004. Proceedings. 20th IEEE International Conference on*, pages 350-359, Chicago Illinois, USA, September, 2004.

[10] Radu Marinescu. Measurement and quality in object-oriented design. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 701-704, Budapest, Hungry, September, 2005.

[11] Stackoverflow - Do you use AOP (Aspect Oriented Programming) in production software? Retrieved August 28, 2014, from http://stackoverflow.com/questions/20663/do-you-use-aop-aspect-oriented-programming-in-production-software

[12] Venkatesh, V., & Bala, H. (2008). Technology acceptance model 3 and a research agenda on interventions. Decision sciences, 39(2), 273-315.