

On Using Interface-types to Break Dependencies

Hayden Melton

April 10, 2006

1 Introduction

It is a widely held belief that the structure of most real software systems does not compare favourably with the instructional literature on software design [FY00, Par96] [SM05, p.84]. In recent works [MT06, MT05] we have concentrated on collecting empirical evidence to support this belief. Particularly we have analysed compilation dependencies among the compilation units of many real, widely-used, open- and closed-source Java applications. In performing this analysis we found that many applications have compilation units (1) involved in long dependency cycles [MT06] and (2) with excessive transitive dependencies [MT05]. These dependency characteristics are contrary to the design principles *avoid dependency cycles among modules* [Par78, Mar96b] [Lak96, p.185] and *prefer a flatter rather than taller module dependency graph* [Lak96, p.196] [Mar96a].

It is often the case that we want to improve the design of poorly-structured code [Fow99, p.55]. The use of an interface-type in place of a concrete-type is the crux of many techniques or strategies that purport to improve structure with respect to compilation dependencies. Examples of such techniques include *Dependency Injection* [Fow04, Fow05], Fowler’s *Extract Interface* refactoring [Fow99, p.341], Martin’s *Dependency Inversion Principle* [Mar96a] and *Interface Segregation Principle* [Mar96c], many of the Gang of Four *Design Patterns* (e.g., Observer) [GHJV95], Lakos’s *Insulation Techniques* [Lak96, ch.6], and Steimann’s advice [SSK03] and patterns [SM05] of interface use.

In this paper we attempt to evaluate the efficacy of these interface-based techniques for breaking cycles and transitive dependencies. We do so from both a structural and conceptual perspective. We conclude that while these techniques are structurally very successful (i.e., they break most cycles and transitive dependencies) in many cases their result is not conceptually sound. That is to say, the resulting structure would be “better” if dependency was broken differently than by introducing an interface-type.

References

- [Fow99] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
- [Fow04] Martin Fowler. Inversion of control containers and the dependency injection pattern. <http://martinfowler.com/articles/injection.html>, 2004.
- [Fow05] Martin Fowler. Inversion of control. <http://www.martinfowler.com/bliki/InversionOfControl.html>, 2005.
- [FY00] Brian Foote and Joseph W. Yoder. Big ball of mud. In N. Harrison, B. Foote, and H. Rohnert, editors, *Pattern Languages of Program Design*, volume 4, pages 654–692. Addison Wesley, 2000.
- [GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

- [Lak96] John Lakos. *Large-scale C++ software design*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1996.
- [Mar96a] R. C. Martin. The Dependency Inversion Principle. *C++ Report*, 8(6):61–66, June 1996.
- [Mar96b] R. C. Martin. Granularity. *C++ Report*, 8(10):57–62, November-December 1996.
- [Mar96c] Robert C. Martin. The Interface Segregation Principle. *The C++ Report*, 1996.
- [MT05] Hayden Melton and Ewan Tempero. A simple metric for package design quality. In *UoA-SE-2005-7*. Department of Computer Science, University of Auckland, 2005.
- [MT06] Hayden Melton and Ewan Tempero. An empirical study of cycles among classes in Java. In *UoA-SE-2006-1*. Department of Computer Science, University of Auckland, 2006.
- [Par78] David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press.
- [Par96] David Lorge Parnas. Why software jewels are rare. *Computer*, 29(2):57–60, 1996.
- [SM05] Friedrich Steimann and Philip Mayer. Patterns of interface-based programming. *Journal of Object Technology*, 4(5):75–94, 2005.
- [SSK03] Friedrich Steimann, Wolf Siberski, and Thomas Kühne. Towards the systematic use of interfaces in java programming. In *PPPJ '03: Proceedings of the 2nd international conference on Principles and practice of programming in Java*, pages 13–17, New York, NY, USA, 2003. Computer Science Press, Inc.