

# Assessing the Effectiveness of Software Modularization Techniques through the Dynamics of Software Evolution

Yuanfang Cai

Department of Computer Science

Drexel University

Philadelphia, PA, 19104

[yfcai@cs.drexel.edu](mailto:yfcai@cs.drexel.edu)

## 1. Introduction

Given the development of new modularization techniques, such as aspect-oriented programming, feature-oriented programming, new assessment techniques are also proposed, such as concern-based metrics [3][4][6] and the metrics based on design structure matrix modeling [2][7] and design rule theory [1]. These metrics assess how concerns are separated in a given software source code or design, or the number of components that are added, deleted or changed between versions.

When a modularization technique is chosen, the designer is making some assumptions about how the software will change. For example, choosing to apply observer pattern implies that the designer assumed that multiple observers will be added later. The software will, almost certainly, be better modularized at the time the modularization technique is applied. However, after the software is changed, it is possible that the software does not change as the designer originally assumed, and the software may suffer from modularity degradation due to this discrepancy. Using a suite of metrics to assess a static software artifact may not be sufficient to assess how well a modularization technique is in terms of accommodating changes.

It seems necessary to develop software metrics that takes the dynamic of software evolution into consideration. We propose to develop a suite of metrics that measure how easily changes are accommodated, e.g. by measuring the average number of modules touched by the modification requests between versions. We call such metrics as *dynamic metrics*. Using these dynamic metrics, the designer can assess whether the software is changing along the way the assumptions were made when major modularization techniques were applied.

Different from existing software metrics that measure on software artifacts such as source code and or design, the dynamic metrics we propose in this paper apply to version histories, that is, the differences between versions. Concretely, we consider the differences in terms of modules added, deleted, and changed, not only between two releases, but also for each change request. For those systems that track changes using existing tools, such as BugZilla, getting the differences caused by modification requests is relatively easy. Otherwise, it is possible to derive the differences from version histories.

Another problem of assessing static software artifacts only is that these metrics will not reflect how well the architecture structure matches the organizational structure. For example, in a globally distributed software project, it is possible that the software structure appears to be well-modularized against a suite of modularity metrics. However, if developers that were remotely located constantly need to communicate to accommodate modification requests, it is possible that there is a mismatch between software architecture and organizational structure that may decrease productivity.

Although the importance of social-technical congruence (STC) has been recognized [8], we still lack an effective metrics to assess STC during software evolution. By measuring dynamic software evolution, it is possible to connect dynamic metrics with STC measurements. For example, we can measure the average number of people who need to contact with to accommodate change requests, as well as the average distances between these developers. The fewer the number of developers need to be contacted with and the closer these developers are located, the better the software architecture is aligned with the organizational structure.

The ultimate purposes of software modularization are to increase software productivity and quality, and to ease software maintenance. Although existing metrics assess the modularity properties of a software at a given time, they do not directly reflect how well the modularization techniques chosen for the software will impact quality,

productivity and maintainability. By assessing software evolution using dynamic metrics, it is possible to link software modularity with quality metrics directly. In the next sections, we will detail the set of dynamic metrics and how to link these metrics with software quality and productivity metrics.

## 2. Dynamic Metrics

In this section, we propose a set of dynamic metrics, working on modification requests. The metrics can be applied to a set of implemented modification requests before releasing the next version.

- *Average Changes (AC)*: for each modification request, we measure how many modules are added, deleted, or changed. This number shows the average from release  $n$  to release  $n+1$ .
- *Average Contacted Personnel (ACP)*: for each modification request, we measure how many developers need to be contacted to accommodate the change. This number shows the average from release  $n$  to release  $n+1$ .
- *Average Change Time (ACT)*: for each modification request, we measure the time needed to accommodate the request. This number shows the average from release  $n$  to release  $n+1$ .
- *Average Distances (AD)*: we can scale the distances between developers according to their relative locations. For example, the distance between co-located team members can be 0, the distance between team members from different cities can be 1, and the distance between team members from different countries can be 2, etc. After that, we can measure the average distances between developers who are involved in accommodating the same modification request. This number will then show the average distances involved to accommodate changes from release  $n$  to release  $n+1$ .

## 3. Measuring Software Outcomes

To measure software productivity, we can use the *number of lines of code (LOC)* developed per person per day, and the *number of resolved change requests* per person per month. For software quality measurement, we can use the *number of defects* found per month per million LOC [5]. Given the dynamic metrics and the software productivity and quality metrics, we can frame a number of hypotheses:

*Hypothesis H1*. In a globally distributed project, the lower *AC*, *ACP*, *ACT* and *AD*, the higher the quality and productivity.

*Hypothesis H2*. If the software is refactored in release  $n$ , and the refactoring activity is successful, then from release  $n$  to release  $n+1$ , average numbers measured by the dynamic metrics should be lower and the quality and productivity value should be higher.

All the metrics and factors can be quantified, and their relations can be statistically calculated. Our next step is to empirically access the applicability of these new dynamic metrics and their effectiveness by testing these hypotheses using large-scale open or close source software projects. For example, we can use these dynamic metrics to measure and compare the impact of different design patterns, system before and after refactoring, or even implementations using different languages.

These metric can also be used to assess the effectiveness of emerging contemporary modularization techniques. For example, to assess and compare an OO vs. AO design, we can employ two groups to implement the same functionality each using different paradigms, and use these dynamic metrics to measure how difficult it is to accommodate changes in each group.

## 4. References

- [1] Carliss Y. Baldwin and Kim B. Clark. *Design Rules, Vol. 1: The Power of Modularity*. The MIT Press, 2000.
- [2] Cai, Y., Huynh, S., and Xie, T. A framework and tool supports for testing modularity of software design. in *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, pages 441–444, November 2007.
- [3] Figueiredo, E. et al.: Evolving software product lines with aspects: An empirical study on design stability. In *Proceedings of the 30th International Conference on Software Engineering*, Leipzig, Germany, 2008.
- [4] Garcia, A. et al. Modularizing design patterns with aspects: a quantitative study. In *Proceedings of the 4th international conference on Aspect-Oriented Software Development (AOSD)*. 2005, pages 3–14.
- [5] MacCormack, A., Kemerer, C.F., Cusumano, M., and Crandall, B. Trade-offs between productivity and quality in selecting software development practices *IEEE Software*. Volume 20, Issue 5, Sept.-Oct. 2003
- [6] C. Sant’Anna et al, “On the modularity of software architectures: A concern-driven measurement framework,” in *Proc of the 1st European Conference on Software Architecture (ECSA)*, Sep. 2007.
- [7] K. Sethi, Y. Cai, S. Wong, A. Garcia, and C. Sant’Anna, “From retrospect to prospect: Assessing modularity and stability from software architecture,” in *Proceedings of the Joint 8th Working IEEE/IFIP Conference on Software Architecture and 3rd European Conference on Software Architecture (WICSA/ECSA) 2009*.
- [8] Cataldo, M., Wangstrom, P.A., Herbsleb, J.D., and Carley, K.M., “Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools,” in *Proceeding of Computer Supported Cooperative Work*, 2006.