

# Assessing the Malleability of Modular Design

Giuseppe Valetto

Drexel University  
Department of Computer Science  
valetto@cs.drexel.edu

## Abstract

One of the main intents of modularization is the creation of a malleable design, one that can easily accommodate change over time. The malleability of a modular structure can be defined operationally in a variety of ways, depending on what development practices or modularization techniques are adopted. We exemplify that by discussing the different meanings of malleable design in agile vs. traditional software development practices, and how it can be differently measured in those two cases. We propose to devise appropriate operational definitions of malleability for the various contemporary modularization techniques, and use them to assess the quality of modular design.

**Categories and Subject Descriptors** D.2.2 [Software Engineering]: Design Tools and Techniques - *Modules and interfaces*. D.2.8 [Software Engineering]: Metrics - *Product Metrics*.

**General Terms** Measurement, Design.

**Keywords** modularization assessment, malleable design, quality of design.

## 1. Introduction

Modularity plays a paramount role in design, as it allows to break down and conquer system complexity, and to assign clear-cut development responsibilities within the development team [6]. Another major benefit of a well-modularized design is its resilience to the unavoidable changes that occur throughout a software project. As remarked in [8], modules and their interfaces provide designers with a set of options for extending and enhancing the software product. Modularity makes design **malleable**, that is, able to sustain change over time, with little or no disruption

to its established intent and structure. Our emphasis on the temporal factor with respect to malleability is intentional. Contrast that with, for example, measures of *design (in)stability* [10], which express the likelihood of a given design to be impacted, whenever some change is required to the system. Malleability, instead, is meant to measure how well software can withstand the “injury of time” (that is, the accumulation of change occurring across its life span), while avoiding the progressive obfuscation and decay of its design

## 2. Position statement

Malleability can be defined in abstract terms as a comparison over a period of time between the amount of change to the modular structure of the system and the amount of change occurring in its implementation. As such, it offers a quantitative perspective on the effectiveness of modularization. This concept of malleability is general, and independent of the mechanisms, practices and technologies used for design and implementation within a project. In operational terms, however, the conceptual definition of malleability must translate into an array of different measures. In particular, whereas implementation changes can be arguably measured generally, for instance via metrics such as code churn [4], changes in modularity are instead quantified differently, depending on the specific modularization technique adopted in a project.

Similarly, malleability may be measured differently, also depending on the adopted development process. As an example, we discuss in the remainder the meanings of malleable design in agile vs. traditional software development, and how different measures are applicable in those two cases. In the discussion that follows, we implicitly assume a “classic” approach to modularization, based upon information hiding. However, analogous considerations could be made for different modularization techniques.

## 3. Planned modularization

For a long time, software design was seen primarily as a planning activity, as design artifacts would represent the

blueprints and the master plan for the software. In that master plan, modularity plays a prominent role, since it enables the assignment along module boundaries of system functionality and development responsibilities. Accordingly, what we could call *planned modularization* ensues, a top-down process of refinement, starting with architectural specifications, and then further breaking down the product into finer-grained elements.

In such a context, a key concern is *conformance* between the planned modularization and its implementation in code. Developers must strive to implement their tasks within the planned modular framework; in turn, each design iteration must strive to reconcile the modular structure with any implementation variation that has remained unaccounted for. Whenever, in the face of substantial implementation work, the system succeeds in remaining conformant to the planned design, we have evidence of a design that's malleable. Huynh et al. [3] propose a technique to automatically enumerate elements of non-conformance within Design Structure Matrices (DSMs) [1] that capture the planned modular structure vs. how it is actually implemented. Following that approach, malleability of a planned modular structure can be measured as **the observed level of conformance, normalized against the amount of code churn in the implementation.**

#### 4. Emergent modularization

With the advent of agile methods, the focus of design activities has shifted. By embracing continuous change and refactoring, agile projects consider design not as a master plan, but as a short-lived, fluid configuration of the software product, which can be modified, as per the stakeholders' needs, at any time across many micro-iterations. In agile methods design is not "dead" [2]; rather, it emerges incrementally from a participatory, evolutionary and possibly implicit process carried forward by all developers from the bottom up. Planned modularization is thus replaced by *emergent modularization*.

In such a context, a key concern is *design convergence*. Continuous refactoring may lead to periods of volatility in the modular structure of the software. Ideally, however, as the project progresses volatility should decrease, and converge towards a high-quality, largely stable design (possibly discounting recurrent oscillations due to design churn [9]). As an emergent modularization becomes incrementally less volatile across agile iterations, it shows increasing levels of malleability in the face of ongoing changes. In this case, malleability is best measured in terms of derivatives, that is, **as the inverse of the rate of design change over time, normalized by the rate of implemented changes, i.e., relative code churn over time, as defined in [5].**

## 5. Conclusions

This paper proposes to assess the quality of modular design in terms of its malleability. Malleability is defined as the resilience of a design to the accumulation of software changes that occur over time in a project.

We maintain that malleability is a generic concept that can be used across the spectrum of modularization techniques, as well as the diverse design approaches embraced by different process models. However, we have shown by example how operational ways to measure malleability may differ in all of those contexts; therefore, finding appropriate measures should become a subject of research.

Further research questions regard the relationships between malleability and other known measures of design (e.g. coupling and cohesion [7], or stability [10]). Malleability could also have interesting consequences onto other qualities of the software product and the development process. In the next future, we plan to investigate the relationship of malleability with fault proneness of a software product, as well as with the effort and cost in a project.

## References

- [1] Baldwin, C.Y., and Clark, K.B. "Design Rules, Vol. 1: The Power of Modularity". The MIT Press (2000).
- [2] Fowler, M. "Is Design Dead?", keynote at the XP Conference 2000, <http://martinfowler.com/articles/designDead.html>
- [3] Huynh, S., Cai, Y., Song, Y., and Sullivan, K. "Automatic Modularity Conformance Checking". In Proc. of ICSE 2008 (2008).
- [4] Munson, J.C., and Elbaum, S. "Code Churn: a Measure for Estimating the Impact of Code Change, in Proceedings of the 14th IEEE International Conference on Software Maintenance (ICSM'98), (1998).
- [5] Nagappan, N., and Ball, T. "Use of Relative Code Churn Measures to Predict System Defect Density". In Proc. of ICSE 2005 (2005).
- [6] Parnas, D.L. "On the criteria to be used in decomposing systems into modules". Comm. of the ACM, 15(12), (1972)
- [7] Stevens, W., Myers, G., Constantine, L. "Structured Design" IBM Systems Journal, 13(2): 115-139(1974).
- [8] Sullivan, K, Cai, Y., Hallen, B, and Griswold, G.W. "The Structure and Value of Modularity in Software Design". In Proceedings of FSE'01 (2001).
- [9] Yassine, A. et al. Information Hiding in Product Development: The Design Churn Effect. Research in Engineering Design, 14:145-161 (2003)
- [10] Yau, S.S., and Collofello, J.S. "Design Stability Measures for Software Maintenance" IEEE Transactions on Software Engineering, 11(9):849-856, September 1985