From Retrospect to Prospect: Assessing Modularity and Stability from Software Architecture

Kanwarpreet Sethi, Yuanfang Cai, Sunny Wong
Department of Computer Science
Drexel University
Philadelphia, PA, USA
{kss33, yfcai, sunny}@cs.drexel.edu

Alessandro Garcia
Informatics Department
Pontifical Catholic University
Rio de Janeiro, Brazil
afgarcia@inf.puc-rio.br

Claudio Sant'Anna
Computer Science Department
Federal University of Bahia
Salvador, Brazil
santanna@dcc.ufba.br

***Abstract*—Architecture-level decisions, directly influenced by environmental factors, are crucial to preserve modularity and stability throughout software development life-cycle. Tradeoffs of modularization alternatives, such as aspect-oriented vs. object-oriented decompositions, thus need to be assessed from architecture models instead of source code. In this paper, we present a suite of architecture-level metrics, taking external factors that drive software changes into consideration and measuring how well an architecture produces independently substitutable modules. We formalize these metrics using logical models to automate quantitative stability and modularity assessment. We evaluate the metrics using eight aspect-oriented and object-oriented releases of a software product-line architecture, driven by a series of heterogeneous changes. By contrasting with an implementation-level analysis, we observe that these metrics can effectively reveal which modularization alternative generates more stable, modular design from high-level models.**

***Keywords*-Design Stability, Metrics, Software Architecture, Software Modularity**

I. Introduction

Development of modular and stable software has been an increasingly abstruse challenge to software architects due to the high volatility of environmental factors [1], [2]. Stakeholder's concerns are classical external factors that influence architecture decisions and, as a consequence, the software modularity and stability [3], [4]. Therefore, the explicit consideration of environmental conditions that drive software changes is a key step to reason about tradeoffs of conventional and modern architecture design alternatives.

Contemporary modularization techniques, such as aspect-oriented (AO) software development [5], have been developed to enhance software modularity and stability, targeting at the production of independently substitutable modules to realize stakeholder's concerns. However, applying aspectual architecture decompositions and comparing with other design alternatives are both far from trivial for many reasons. First, it has been recognized that AO decompositions do not always guarantee better modularized [6] and stable [1], [3] software. Second, systematic comparisons of contemporary modularization techniques are usually based on source code analysis [1], [3], [7]. More fundamentally, conventional metrics for assessing modularity and stability of software architecture decompositions (e.g. [2], [8], [9]), do not take into consideration external factors that drive software changes.

In this paper, we present a novel set of architectural-level software modularity and stability metrics. Our metrics are different from previous metrics in two aspects. First, these metrics embrace environmental conditions as part of the assessment criteria. The rationale is that besides the internal structure of software, its stability is also closely associated with external factors that drive software changes [4], [10]. Second, our metrics measure how well a design structure can generate independently substitutable modules, e.g. *options* [4], [11]: an independently substitutable module provides the user an option to swap out the module with a better version or to keep the current version if it remains to be the best [4], [11]. These metrics can be formally defined, hence automatically calculated, based on an emerging *design structure matrix* (DSM) model [11] and its formal counterpart, the *augmented constraint network* (ACN) [12], [13].

We evaluate these metrics to see whether designers can confidently compare design alternatives without implementation details. We apply them to eight releases of a software product line (SPL) for handling multimedia on mobile devices, called MobileMedia [3], [14]. Both aspect-oriented (AO) and object-oriented (OO) editions of MobileMedia were used and compared. Modularity and stability were key requirements in both editions, which underwent changes as specified in [3]. We compared the conclusions obtained from our assessment with the conclusions obtained from in-depth analyses previously made from the source code [3]. The results showed that our approach reached highly consistent conclusions against that of implementation-level analysis, revealed several problems in previous implementation-level analysis, and led to new insights.

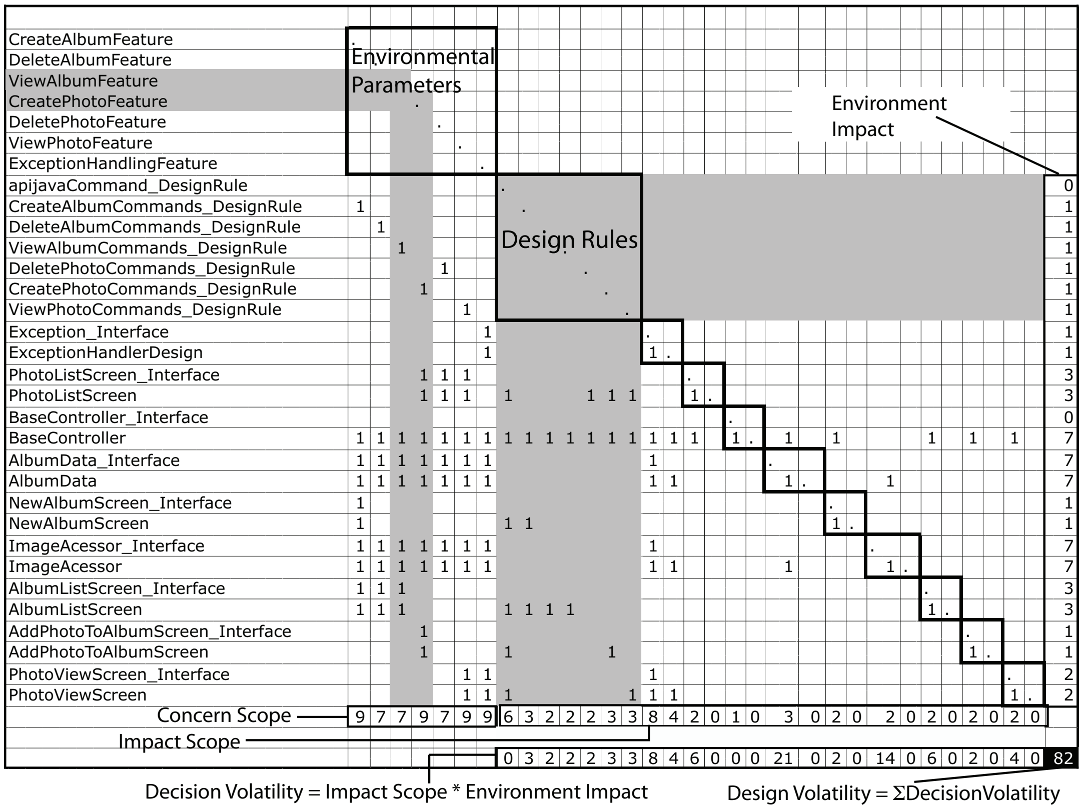

Figure 1. DSM with Metrics for Release 2 of MobileMedia OO Design

II. Stability and Modularity Metrics

This section presents a suite of design stability and modularity metrics that explicitly take environmental conditions and option reasoning into consideration. These metrics can be uniformly and formally defined base on an *augmented constraint network* (ACN) [12], [13], in which both design dimensions and environmental conditions are modeled as variables and their relations are modeled as logical constraints. From an ACN, a *design structure matrix* (DSM) can be automatically generated: the columns and rows of the square matrix are labeled with design dimensions and the cells are marked using a pair-wise dependency relation automatically derived from the ACN [12]. We use the DSM of the second release of MobileMedia (Figure 1), derived from an ACN transformed from the corresponding UML component diagram, to explain our metrics. The first block of the DSM contains 7 variables, modeling all the features as environmental variables. The second block contains *design rules* [11], such as Java Command API and event tags.

Design Stability Metrics. Software stability is usually measured based on its coupling structure. However, it is possible that some part of the system is highly coupled but is not subject to any environmental changes, resulting a stable design with a low stability value. We thus propose a $Decision\ Volatiltiy$ metric that assesses the stability of *a decision* in terms of the number of environmental conditions that influence it ($Envr\ Impact$), and its own impact scope ($Impact\ Scope$). The rationale is that the more environmental conditions influence it, the more likely it will be subject to change; the more decisions it can influence, the more impact it can have on the stability of the whole design. The $Design\ Volatility$ is thus the summation of all individual $Decision\ Volatiltiy$ values. Figure 1 shows the MobileMedia OO Release 2 DSM in which variables are clustered into *environment*, *design rule*, and component blocks (shown as blocks consisting of 2 variables). This DSM visually shows how the volatility metrics can be calculated.

Modularity Metrics. The better a system is modularized, the better concerns should be localized and separated from each other, and the easier a module can be swapped with a better version. After modeling concerns as environmental variables, its $Impact\ Scope$ value can be used to measure how the concern is diffused, hence the $Concern\ Scope$ metric. Figure 1 shows the $Concern\ Scope$ values for the 7 features in the second release of MobileMedia. The lower the number, the lesser the concern is diffused.

The $Concern\ Overlap$ metric quantifies how two concerns interact with each other: the more decisions shared by two concerns, the greater the overlapping. In Figure 1, the concern scopes of *ViewAlbumFeature* and *CreatePhotoFeature* are lightly shaded to highlight that they share 5 variables, so their $Concern\ Overlap$ value is 5.

The $Independence\ Level$ (IL) metric is proposed to quantify the extent to which a design can support module-wise independent searching and replacement, that is, its ability to generate option values. The algorithm we use to discover independent modules can be found in [15], which clusters all the variables of an ACN/DSM into a *design rule hierarchy* (DRH) exemplified by the DSM shown in Figure 2. The decision in any cluster only make assumptions about the decisions in previous clusters. The only truly independent modules are the clusters within the lower-right block. We define a simplified option-oriented *Independence Level* metric as the percentage of the system that can be freely and independently swapped or evolved under stable design rules. The rationale is that the more variables in independent modules, the larger part of the system can evolve independently. For example, the $Independence\ Level$ of the second release MobileMedia OO architecture is 0.28, meaning that about 28% of the system consists of independent modules and can effectively generate option value.

		2	3	4	5	6	7	8	9	10	11	12
Level0	2	3										
Level1	3	33	0									
NewAlbumScreen	4	2	2									
PhotoListScreen	5	4	4		.							
PhotoViewScreen	6	5	2			.						
AddPhotoToAlbumScreen	7	2	2				.					
AlbumListScreen	8	4	4					.				
ImageAcessor	9	9	2						.			
BaseController_Interface	10	0	0							.		
BaseController	11	10	12							1	.	
AlbumData	12	9	2									.
			Release 2									

Figure 2. DSM Clustered with DR Hierarchy, OO R2 MobileMedia Architecture

III. Evaluation

To evaluate these metrics, we compare the conclusions reached by our approach with the conclusions obtained from previous implementation-level analysis conducted by Figueiredo et al. [3], and investigate the reasons behind each discrepancy. Each release of MobileMedia [3] evolves from the previous release by adding some new features or restructuring the previous version to achieve a better modularized structure. Features added can be *mandatory*—added to the core functionality and are applicable to all devices and media types, *optional*—elective based on the API support of each device, or *alternative*—the various media types (video, music, photo) supported by the device.

We first built a tool to automatically generate basic ACN models from the component diagrams recovered by Figueiredo et al. for both versions of each edition. Then we complement ACN construction with environmental parameters, the features, and implicit design rules. Finally, we compute the stability and modularity properties for all 16 ACNs, and compare our findings with previous source code analysis. The smallest ACN (OO release 1) has 29 variables and the largest ACN (AO release 8) has 88 variables. For each release, the OO and AO ACNs share the same set of environmental parameters.

Stability Analysis Results. Our architectural-level volatility metrics show that AO designs always appear to be more unstable than OO designs, implying features are more diffused and/or components have higher impact scopes on average (higher coupling between components) for AO architectures. Our conclusion is exactly the same as that of Figueiredo et al.'s. Their $coupling$ and $lack\ of\ cohesion$ measurements show that AO implementations suffer from higher coupling and lower cohesion due to a side effect of the aspectization of alternative and optional features.

Modularity Analysis Results. For mandatory features, our architectural-level $Concern\ Scope$ assessment shows highly consistent results with that of implementation-level $Separation\ of\ Features$ analysis: AO architecture is generally harmful as all the concerns, that is, environmental parameters, influence more dimensions for all the releases because a change in a mandatory feature could potentially alter these newly added aspects that depend on the core components implementing the mandatory features, increasing its diffusion.

The only discrepancy between the assessments in the mandatory category is for the *ExceptionHandling* feature. Although Figueiredo et al. [3] put this feature into the category where AO wins because AO employed fewer number of lines of code (LOC), their assessment also shows that the *ExceptionHandling* feature diffuses to more components and shows higher level of coupling in AO versions. Their conclusion is that AO was "not very stable", which is consistent with our conclusion.

For alternative features, both our architecture-level assessment and their analysis show that AO and OO have similar effects. For optional feature comparison, there are two major discrepancies. Our architecture-level assessment shows that AO architecture is superior for all optional features, while their implementation-level assessment reports that both paradigms have similar effects for *SMS* and *CopyMedia* features. After investigating the differences, we found that our analysis is more accurate in that ACN modeling picks up indirect dependencies that were missed from implementation-level analysis. For example, we approach detected that the *CopyMedia* Feature in OO design to be indirectly diffused over three controllers through GUI.

We report the $Concern\ Overlap$ measurements for two pairs of features to make the results comparable with that of implementation-level assessment: *Controller* vs. *Labeling* and *Sorting* vs. *Labeling*. Conclusions from both assessments are exactly the same: for the first pair, the AO architecture generates higher degree of overlapping; for the second pair, the AO architecture generates less overlapping because the optional feature of *Sorting* is aspectized and has less points of impact from mandatory features.

Our $Independence\ Level$ metric shows that the AO architecture's option-generation ability is lower than that of OO for the *Exceptionhandling* feature, and its ability increases much faster for optional features than OO versions, but decreases for alternative features because newly added aspect modules have to depend on mandatory core components, so that these core modules are not independent anymore, offsetting the overall option-generation ability. The AO IL values are generally higher than that of OO, showing that higher volatility does not imply inferior option-generation ability.

IV. Related Work

Several metrics have been proposed in the past years to assess software design modularity and stability. One of the most referenced suites of metrics is the one proposed by Chidamber and Kemerer [16]. These metrics quantify source code modularity using coupling, cohesion and interface size. Our work differs because our metrics work on a higher-level, use more abstract architecture models, and take external factors and module substitutability into consideration.

Concern-oriented metrics [3], [17] are proposed to explicitly measure concern scattering and overlapping (e.g. features) from which software stability can be inferred. Our stability metrics similarly integrate concerns, but can be directly calculated from ACN models. Briand et al's work [18] measures maintainability from high-level design models using metrics of coupling, cohesion and variability, but not considering external factors or module independency as our work does.

V. Conclusions and Future Work

This paper presents a suite of architecture-level metrics and showed the possibility of assessing AO vs. OO alternatives without implementation details. The metrics add value to the state-of-the-art by allowing designers to analyze the option-generation ability of an architecture, and to assess design stability taking into account the impact of external environmental conditions. Our future work includes automatically derive environmental variables and design rules from requirement documents and major structural patterns. We also plan to conduct a pro-active case study involving on-going real software projects to assess the predictive ability of these metrics. Some of the metrics presented in this paper are simplified and can be further extended as needed. For example, the volatility metrics can be extended with the probability of change for each environmental condition and be used to conduct sensitivity analysis to assess design stability under uncertainty.

References

[1] P. Greenwood et al, "On the impact of aspectual decompositions on design stability: An empirical study," in *Proc. of the 21st ECOOP*, 2007.

[2] S. Yau and J. S. Collofello, "Design stability measures for software maintenance," *IEEE Trans. Softw. Eng.*, vol. 11, no. 9, pp. 849–856, 1985.

[3] E. Figueiredo et al, "Evolving software product lines with aspects: An empirical study on design stability," in *Proc. of the 30th ICSE*, May 2008.

[4] K. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen, "The structure and value of modularity in software design," in *Proc. of the 8th FSE*, Sep. 2001, pp. 99–108.

[5] Robert E. Filman et al., *Aspect-Oriented Software Development*. Boston: Addison-Wesley, 2005.

[6] K. Sullivan et al, "Information hiding interfaces for aspect-oriented design," in *Proc. of the 10th FSE*, Sep. 2005, pp. 166–175.

[7] K. Hoffman and P. Eugster, "Towards reusable components with aspects: an empirical study on modularity and obliviousness," in *Proc. of the 30th ICSE*, 2008, pp. 91–100.

[8] A. Molesini, A. F. Garcia, C. von Flach G. Chavez, and T. V. Batista, "On the quantitative analysis of architecture stability in aspectual decompositions," in *Proc. of 7th WICSA*, 2008, pp. 29–38.

[9] D. Kelly, "A study of design characteristics in evolving software using stability as a criterion," *IEEE Trans. on Soft. Eng.*, vol. 32, no. 5, pp. 315–329, 2006.

[10] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–8, Dec. 1972.

[11] C. Y. Baldwin and K. B. Clark, *Design Rules, Vol. 1: The Power of Modularity*. MIT Press, 2000.

[12] Y. Cai and K. Sullivan, "Simon: A tool for logical design space modeling and analysis," in *Proc. of the 20th IEEE/ACM ASE*, Nov. 2005.

[13] Y. Cai, "Modularity in design: Formal modeling and automated analysis," Ph.D. dissertation, University of Virginia, Aug. 2006.

[14] T. Young, "Using aspectj to build a software product line for mobile devices," Master's thesis, University of British Columbia, Aug. 2005.

[15] S. Huynh, Y. Cai, and K. Sethi, "Design rule hierarchy and analytical decision model transformation," Drexel University, Tech. Rep. DU-CS-08-04, Nov. 2008, https://www.cs.drexel.edu/content/uploads/Research/DU-CS-08-04.pdf.

[16] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. on Soft. Eng.*, vol. 20, no. 6, pp. 476–493, 1994.

[17] C. Sant'Anna et al, "On the modularity of software architectures: A concern-driven measurement framework," in *Proc of the 1st ECSA*, Sep. 2007.

[18] S. M. L. Briand and V. Basili, "Measuring and assessing maintainability at the end of high level design," in *Proc. of the 9th ICSM*, Sep. 1993.