

Assessing Design Modularity and Stability using Analytical Decision Models

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

Alessandro Garcia, Claudio Sant'Anna
Computing Department
Lancaster University
Lancaster, United Kingdom
{garciaa, c.santanna}@comp.lancs.ac.uk

Abstract

Various new modularization techniques, such as aspect-oriented (AO) programming, are proposed to improve software stability and modularity, and there is a pressing need to assess tradeoffs of modularization alternatives from high-level design models instead of source code. We present the following contributions in this context: we first present a technique to automatically transform a UML component diagram into analytical decision models, the augmented constraint network (ACN) and design structure matrix (DSM). Second, we define and formalize a suite of design stability and modularity metrics, based on ACN and DSM modeling. These metrics improve prevailing metrics by taking environmental conditions and option reasoning into consideration. Finally, we evaluate our techniques using eight OO and AO releases of a software product line. We show that these metrics allow us to assess which technique generates a more stable, modular design, given a series of envisioned changes, verified by previous source code analysis.

1. Introduction

Development of stable software designs has increasingly been a deep challenge to software architects [2] due to the high volatility of their concerns and associated design decisions [20, 27]. Design stability is directly dependent on the ability of a software architecture to sustain its modularity properties and not succumb to modifications [27, 3]. Recent empirical studies have also pointed out that design instabilities are directly proportional to the degree of concerns scattering and overlaps [12, 20]. In order to enhance design modularity and stability, software designers are increasingly being equipped with contemporary modularization techniques, such as aspect-oriented (AO) software development. However, new modularity mechanisms significantly increase the space of design decisions, thereby complicating the achievement of stable decomposition choices from early

design stages. Even worse, it has also been recognized that the use of AO design mechanisms do not always guarantee better modularized [25, 16] and stable [12, 20] software.

These issues make the assessment of alternative AO and non-AO software decompositions difficult, especially at the architecture conception stage when the most crucial design decisions are taken. To better understand which modularization technique is better under what circumstances, researchers have conducted systematic assessments strictly based on source code analysis [12, 14, 17, 20, 13]. These studies have derived preliminary cookbooks of best practices with AO mechanisms in terms of low-level code idioms [14, 1, 13, 20, 21]. However, inferring design modularity and stability after investing in implementations can be expensive and impractical. In addition, with the transition to model-driven software engineering gaining momentum, assessments of alternative architectures need to be undertaken from high-level design representations, such as UML-based component models [19].

Another obstacle comes from the fact that existing metrics for AO decompositions (e.g. [10, 6, 29]) do not rely on unified architecture design representations, making the comparison with OO decompositions cumbersome and hard to scale [9]. Moreover, according to Baldwin and Clark's influential *design rule(DR) theory* [3], a better-modularized design should allow for more independent modules that can generate higher option values [24, 17, 8]. The existing metrics, such as coupling and cohesion, are related to this principle but do not directly measure the ability of a design to generate options [3].

Our claim in this paper is that the *design structure matrix* (DSM) [23], along with the *augmented constraint network* (ACN) [7] - that formalizes the DR theory - have the potential to serve as unified high-level design models for modularity and stability assessment. Our concrete contributions are as follows. First, we present an approach that automatically transforms UML component diagrams to ACNs in order to compute design modularity and stability properties (Section 3). DSMs can be derived from such ACNs to vi-

sualize design modular structure. We then complement the automatically generated ACNs with environmental parameters and implicit design rules that are not explicit in a UML design model.

Second, we formally define a suite of ACN/DSM-based metrics that support the assessment of design alternatives from multiple modularity and stability perspectives (Section 4). The following stability metrics are formalized: 1) the *Design Volatility* metric that measures design (in)stability under the effect of changing environmental parameters; 2) the *Concern Scope* metric that measures the impact of each concern scattering on the design stability; 3) the *Concern Overlap* metric that estimates stability in terms of how two concerns are intermingled with each other; and 4) the *Change Impact* metric that measures design space expansion, contraction and modification during software evolution. To measure the option-creation ability of a design, we propose a *Independence Level* metric to estimate to what extent a design can support independently-evolvable modules. We create this metric to support *option* reasoning proposed by Baldwin and Clark [3].

Finally, we evaluated the effectiveness of the proposed metrics by applying them to a series of architecture releases of a software product line (SPL) for handling multimedia on mobile devices, called MobileMedia [12](Section 5 and 6). In order to assess the effectiveness of our proposal, we contrast our findings to an in-depth analysis previously made from the source code level [12]. We show that the new assessment approach leads to consistent conclusions. The implication is that it appears possible to reasonably assess stability and modularity of design alternatives without digging into implementation intricacies.

2. Background

This section briefly introduces the key concepts that form the basis of our assessment approach: the *design structure matrix* (DSM) modeling, *design rule* (DR) theory, and the *augmented constraint network* (ACN), a logic-based design modeling technique.

Design Structure Matrix and Design Rule Theory A DSM is a square matrix in which the columns and rows are labeled with design variables, and a marked cell models that the decision on the row depends on the decision on the column [23]. Figure 1 depicts a DSM modeling one of the MobileMedia releases. The blocks along the diagonal model the modules in the system. According to Baldwin and Clark, *design rules* (DRs) are stable decisions that decouple otherwise coupled decisions and thus create *independent modules* [3], modeled through asymmetric dependencies in a DSM. A module is *independent* if it depends only on design rules, but not on other independent modules. In Figure 1, the blocks in zone 8 are indepen-

dent modules. The independence between modules, created by design rules, is a key modularity property. Baldwin and Clark proposed a *net option valuation* (NOV) model [3] that quantitatively accounts for how a modular structure generates option values. Intuitively, modularity adds value in the form of real options—a module creates an option to invest in a search for a superior replacement, or to keep the current one if it is still the best choice.

Augmented Constraint Network. Cai et al [7] have developed a logic-based design modeling technique, called the *augmented constraint network* (ACN) that formalizes the DSM modeling and design rule theory. An ACN consists of a *constraint network* (CN) that models the logical relations among design decisions, a *dominance relation* (DR) that formalizes the notion of design rules and the asymmetric dependencies among design decisions, and a *cluster set* (CS) in which each cluster models one way a design can be aggregated into modules, similar to the different ways the rows and columns of a DSM can be reordered. A constraint network consists of a set of *design variables* (Var), their *domains* (Dom), and a set of *constraints* (Con). Formally: $ACN = \langle CN, DR, CS \rangle$ and $CN = \langle Var, Dom, Con \rangle$.

Similar to DSM modeling, ACN modeling entails paradigm-agnostic means to compute modularity properties of a design, and provides precise definition for the *pair-wise dependence relation* (PWDR): if a design variable y depends on variable x , that is, $(x, y) \in PWDR$, then there must exist a consistent state of the ACN. Changing the value of x will violate some constraints and make the constraint network inconsistent, and the value of y has to be changed in one of the ways to restore the consistency. A DSM can be thus automatically generated: the computed PWDR is used to populate the matrix and a selected cluster is used to order the columns and rows of the DSM. Both DSMs in Figure 1 and 3 are automatically generated from the same ACN using different ways of clustering.

3. Model Transformation

In this section, we illustrate how to derive an ACN model starting from a UML component diagram. The ACN model forms basis for automatic modularity and stability calculation. As mentioned in Section 1, MobileMedia is a software product line for mobile devices supporting photo, music and video features. It has been used in a previous case study on design stability from the source code [12]. Figure 2 shows the component diagram of the fourth release of object-oriented MobileMedia design, which we use as a running example.

Formalized Component Diagram A UML component diagram has the following elements: a rectangular box with a component symbol representing a component in the diagram, a lollipop symbol connected to a component rect-

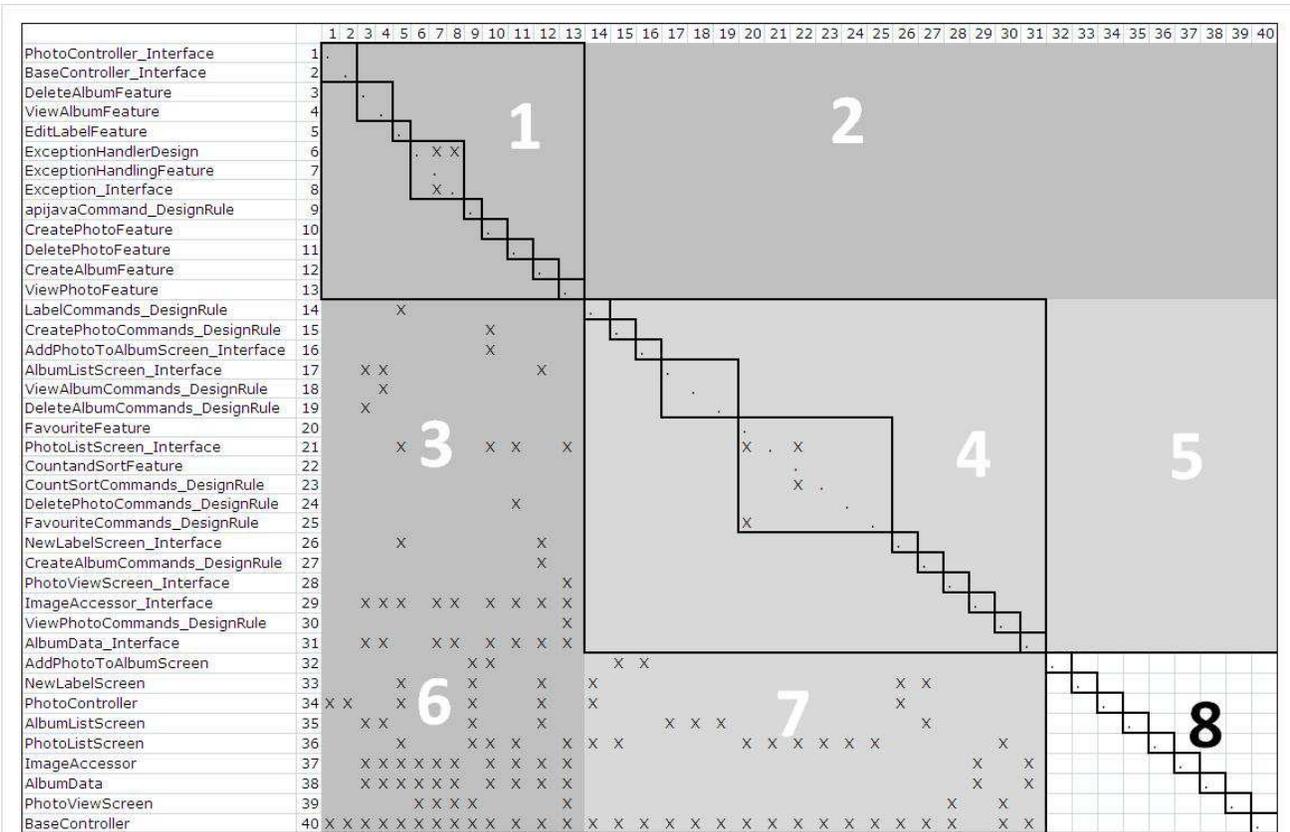


Figure 1. Complete Design Structure Matrix for the MobileMedia Object Oriented Release 4

angle representing an interface *provided* by the component, and a socket symbol connecting to an interface lollipop of another component representing an interface *required* by the component. Currently UML modeling does not support modeling of aspects. Therefore our diagrams define aspects similar to components. A rectangular box without a component symbol represents an aspect. The aspectual pointcuts are realized using a dashed line connected to interfaces.

ACN Transformation Given the representation of a component diagram, we transform it into an ACN, that is, a set of variables, their constraints and dominance relation, as follows. A component element is modeled using a variable named as `nameOfComponent`, and an interface is modeled as a variable named as, `nameOfComponent + _Interface`. An aspect element is modeled as a variable named as, `nameofAspect + _Aspect` We assign each variable at least two values, `orig` and `other`, to model the current design (`orig`) and a possible change (`other`). For example, the `AlbumData` component and one of its interfaces are modeled as `AlbumData_Interface: {orig, other}`, and `AlbumData: {orig, other}`. Next we translate the relations between interfaces and components as logical constraints. For example, the `AlbumListScreen` com-

ponent requires `BaseController_Interface` and is translated into the following constraint: `AlbumListScreen = orig => BaseController_Interface = orig`.

The dominance relation is generated as follows: every interface that is provided or required by a component, dominates that particular component. This means that a change in the interface may propagate to the component, but a change in the component can not propagate to the interface. The following sample pairs are the members of the dominance relation of the automatically generated ACN:

```
(AlbumListScreen, BaseController_Interface);
(AlbumListScreen, AlbumListScreen_Interface);
```

Complementing with Environmental Conditions and Design Rules The automatically generated variables, constraints, and dominance relation, plus a random cluster, constitute a basic ACN model. This is far from sufficient, because a component diagram does not model environmental conditions and important but implicit design rules. We introduce how the environmental conditions and design rules were identified in MobileMedia in Section 5.

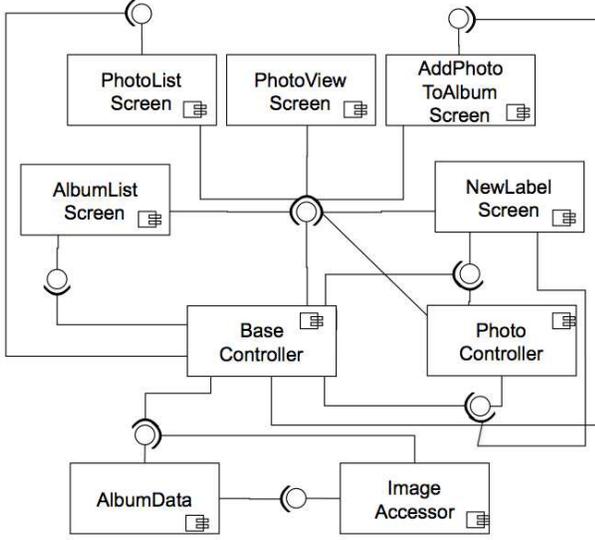


Figure 2. UML Component Diagram

4. Stability and Modularity Metrics

This section presents a set of stability metrics and a modularity metric that are formally defined based on DR/DSM/ACN modeling. The stability metrics are the design-level counterpart of the metrics defined over source code structures [12], and the new modularity metric is simplified from Baldwin and Clark’s option theory to avoid complicated NOV parameter estimation that software engineers are usually not familiar with. We show that these metrics can be uniformly and formally defined, and hence be automatically measured, in an ACN model.

4.1 Stability Metrics

Design Volatility Metrics Software stability is usually measured based on how software components depend on each other syntactically, such as the number of classes outside a package that depend on classes within the package [27]. Large number of empirical studies have been conducted to assess how modern programming paradigms influence software stability [12, 20]. The issue is that traditional volatility metrics do not explicitly take environmental conditions into consideration. For example, it is possible that some part of the system is highly coupled but is not subject to environmental changes. As a result, the design could have a high volatility value, but is highly stable.

We thus propose a *DecisionVolatility* metric to measure the stability of a design decision, x , and a *DesignVolatility* metric to measure the stability of the whole design, d . The *DecisionVolatility* metric assesses the stability of a decision in terms of the number of environ-

mental conditions that will influence it (*EnvrImpact*), and its own impact scope (*ImpactScope*). 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, its volatility will have more impact on the stability of the whole design. The *DesignVolatility* is thus the summation of all *DecisionVolatility* values.

Sullivan et al. [24] first introduced *environmental parameters* into software DSMs, and an ACN can similarly model the environmental parameters as variables, and all the environmental variables can be aggregated into a special cluster, *envr*. From the pair-wise dependence relation *PWDR* derived from the ACN, with variable set *var*, we can formally define these metrics:

$$EnvrImpact(x) = |\{e \in envr \mid (x, e) \in PWDR\}|$$

$$ImpactScope(x) = |\{y \mid (x, y) \in PWDR\}|$$

$$DecisionVolatility(x) =$$

$$EnvrImpact(x) \times ImpactScope(x)$$

$$DesignVolatility(d) =$$

$$\sum_{x \in var \setminus envr} DecisionVolatility(x)$$

Figure 3 shows the MobileMedia OO release 4 DSM in which the environmental parameters are clustered into the first block to distinguish from other design variables. This DSM visually shows how the volatility metrics can be calculated from the PWDR relation derived from an ACN model: the numbers in the column to the right of the DSM are the total number of environmental variables that influence the variable on the corresponding row. The numbers in the row next to the last row of the DSM are the impact scope of the variable on the column. The numbers below that are the *DecisionVolatility* value of each variable, which sum to the design volatility value shown in the cell with dark background and white text.

Figure 3 shows that the volatility of the whole design is 99. From the *DecisionVolatility* values, we can identify the most unstable variable. In this case, *AlbumDataInterface* seems to be most volatile decision. Although it only influences three other variables, it turned out to be highly influenced by environmental change. The observation is confirmed by the designer: whenever a feature is added, deleted or changed, in terms of photo, music or video, the data set needed to be accessed will have to be changed, and hence the change of the interface.

Concern Scope Metric Concern diffusion or scope is usually measured in source code by counting which components, operations, or lines of code are related to a particular concern. Based on ACN/DSM modeling, a concern, a feature, or a requirement, can be modeled as an environmental variable, and its impact scope can be used to measure how the concern is diffused. For example, according to the DSM shown in Figure 3, there are 10 features in the fourth release of MobileMedia. (Note: In this paper, we use the term concern and feature interchangeably since features are the only

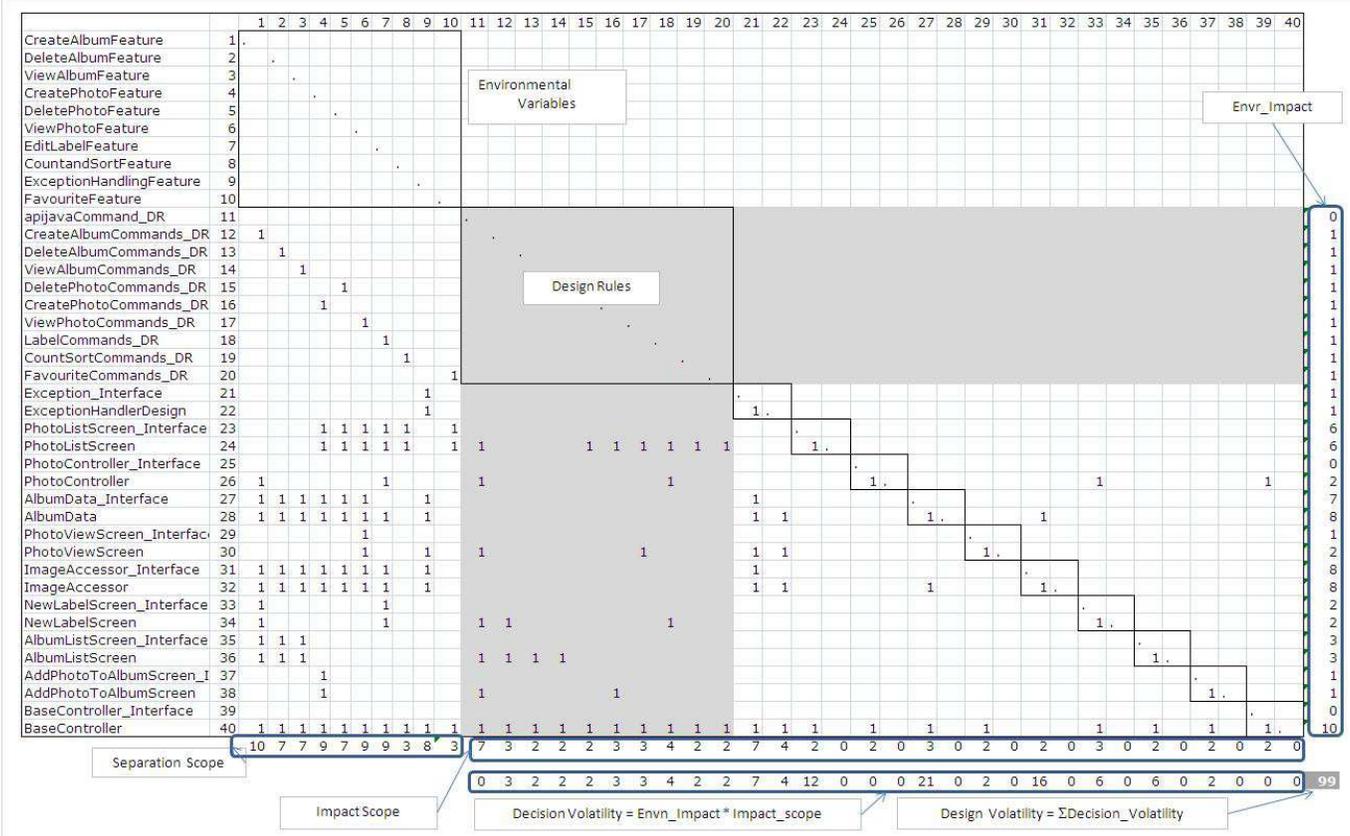


Figure 3. Complete Design Structure Matrix for MobileMedia Object Oriented Release 4

concerns we model) The impact scopes of these concerns are calculated by the number of variables influenced by the feature. For example, the `FavouriteFeature` influences 3 variables, meaning that this concern is spread into 3 decisions. The fewer the number, the lesser the concern is diffused. For example, the `CreateAlbumFeature` concern has a greater impact scope than the `FavouriteFeature` because 10 decisions are involved in the concern.

We formally define the scope of a concern, c , as the number of variables that depend on or influence c , which can be calculated from the PWDR derived from the ACN:

$$ConcernScope(c) = |\{v \mid (v, c) \in PWDR \vee (c, v) \in PWDR\}|$$

Concern Overlap Metric This metric measures how two features interact with each other. Intuitively, if a variable is influenced by two features, then these two features overlap over this variable. A greater overlapping of concerns, means the scope of two concerns is tangled and hence might affect stability. For example, the `PhotoController` variable is influenced by both `EditLabelFeature` and `CountandSortFeature`, which means that these two features overlap on this variable. We can compute how two features (concerns), c_1 and c_2 , overlap according to the fol-

lowing formalization:

$$ConcernOverlap(c_1, c_2) = |\{v \mid (v, c_1) \in PWDR \vee (c_1, v) \in PWDR\} \cap \{v \mid (v, c_2) \in PWDR \vee (c_2, v) \in PWDR\}|$$

Change Impact Metric By comparing the number of variables that are added, removed or changed in two ACNs, acn_i and acn_{i+1} , that model consecutive design evolution, we can define a vector to analyze change impact:

$$ChangeImpact(acn_{i+1}, acn_i) = \langle addition, removal, change \rangle$$

where

$$\begin{aligned} addition &= |\{v \mid v \in acn_{i+1} \wedge v \notin acn_i\}| \\ removal &= |\{v \mid v \in acn_i \wedge v \notin acn_{i+1}\}| \\ change &= |\{v \in var \mid v[acn_i] \neq v[acn_{i+1}]\}| \end{aligned}$$

4.2 Modularity Metrics

Instead of measuring design modularity in terms of coupling and cohesion, we propose to measure to what extent a design can support module-wise independent searching and replacement, that is, its ability to generate option values. Intuitively, assuming that the design rules are stable, the more independent modules in the system, the higher option value

can be generated.

Baldwin and Clark proposed a sophisticated *net option valuation (NOV)* model to statistically account for the options values that can be generated by a modular structure. Directly measuring NOV values from a design model has two obstacles. First, the NOV formula requires a number of assumptions that are usually not readily measurable, such as the technical potential of each module. Second, and more fundamentally, the definition of *modules* in Baldwin and Clark’s theory, consistent with Parnas’s widely-recognized definition, is independent task assignment. But this is not the definition used by prevailing design models that use classes, aspects, or components as modules, which are usually not *independent*. We need to identify truly independent modules first.

Our recent work presents [15] an algorithm to define and discover these independent modules, and to cluster a DSM into a *design rule hierarchy (DRH)* that can reveal the assumption structure within large-scale software system. Figure 1 shows the automatically DRH clustered DSM for the design of the fourth release (OO version) of the MobileMedia system. It shows that the design rule hierarchy has three layers. The right-lower block (zone 8) contains all the clusters that are independent modules that depend on the variables preceding them. The block in zone 4 contains clusters of decisions that only make assumptions about the decisions in zone 1, and influence decisions in zone 8. Moreover, once the decisions in zone 1 are made, the clusters of decisions in zone 4 can be made independently and concurrently. The only truly independent modules, as shown in the DSM, are the clusters in zone 8. All the other variables are environmental parameters or design rules that make these modules independent from each other.

We thus define a simplified option-oriented *Independence Level* metric based on design rule hierarchy: the percentage of the system that can be freely changed under stable design rules. The rationale is that the more variables in independent modules, the larger part of the system can evolve independently.

We define the independent module set of an ACN, A , computed from our *design rule hierarchy* algorithm, as the *IndependentClusters*, and define the *Independence Level* of the design as the number of variables in *IndependentClusters* relative to the total number of variables, var , of the ACN:

$$IndependenceLevel(A) = \frac{|\{v \mid \exists i \in IndependentClusters v \in i\}|}{|var|}$$

For example, the *Independence Level* of the OO design for the 4th release of the MobileMedia, whose DSM is shown in Figure 1 and 3, is 0.23. This number means that only about 23% of the system can effectively generate option value.

5. Evaluation

Our goal is to evaluate to what extent the ACN-based metrics can faithfully assess stability and modularity properties from high-level design models. The main strategy is to compare the assessment results obtained from high-level models with previous source code analysis results [12]. This section describes the subject system, the evaluation procedures and results.

5.1 Subject Systems

We evaluate our approaches using a Software Product Line (SPL) called MobileMedia [28] which is architected both using object-oriented and aspect-oriented techniques. MobileMedia provides support for handling photo, video and music data on mobile devices, such as cellular phones. Eight releases of both aspect-oriented and object-oriented designs were implemented [12] based on an original release developed at the University of British Columbia.

Each release evolves from the previous release by adding some new functionalities or restructures the previous version to achieve a better modularized structure. Release 1 of MobileMedia is the core design that only includes photo-related core functions, such as adding or deleting photos or photo albums. Release 2 added exception handling as a mandatory feature. Release 3 added photo labeling as a mandatory feature and photo sorting as an optional feature. Releases 4, 5, 6 each added an optional feature, such as specifying favourite photos. Release 7 involved changing the mandatory characteristic of photo manipulations to become an alternative feature, and adding another alternative feature, music manipulation. The final release 8 added another alternative feature, video manipulation. Mandatory features are defined as features adding to the core functionality and are applicable to all mobile devices running MobileMedia. Optional features are elective based on the API support of each device. Alternative features are the various media types (video, music, photo) supported by the device.

Researchers have recently analyzed the source code of MobileMedia for each release in great detail [12], and compared the tradeoffs of each OO and AO implementation alternative in terms of stability and modularity properties, such as concern diffusion, change impact analysis, coupling and cohesion. The component diagrams are available for both versions of each release, providing the starting point of our analysis.

5.2 Assessment Procedures

Our evaluation was divided into the following stages: (1) the automatic generation of basic ACN models from component diagrams as introduced in Section 3; (2) comple-

mentary ACN construction with environmental parameters and implicit design rules; (3) computation of the stability and modularity properties for each ACN using the metrics introduced in Section 4; (4) comparison of our findings with previous source code analysis.

Basic ACN Model Generation The component diagrams we used for MobileMedia deviate slightly from the UML standard because new elements were defined and added to model pointcuts and aspects in AO versions. But these diagrams maintained by domain experts have a unified textual representation. We developed a small program that takes the textual representation of component diagrams as input, and converts them into 16 ACN models.

Complementary ACN Construction The automatically-generated ACN models do not contain environmental parameters and implicit architectural design rules. We identify them and complement the basic ACNs as follows. There exist large number of environmental conditions that are potentially relevant to MobileMedia, such as the features needed and the hardware constraints of devices. To make the results comparable with previous source code analysis results, we only model features as environmental conditions. As shown in Table 1, there are 6 features in the 1st release and 15 features in the 8th release.

Depending on model granularity, the number of implicit design rules can similarly be non-trivial. Since we aim at architecture assessment, we only model high-level architectural design rules, many of which are implicit in MobileMedia. For example, the overall MobileMedia event handling system is designed with the Observer pattern. In this pattern, the subjects are a subset of the components and the observers are various controllers. The pattern uses Java Command API as the communication framework so that the components are decoupled from the controllers. We thus add a new design rule called `apijavaCommand_DesignRule` to model the Java API and its decoupling effects. Another critical design rule is that when a component raises an event, it passes along a tag as one of the event parameters to identify itself, such as “add photo”. This tag is caught by controllers to respond to the event. As a result, a component only knows the tag and the Java API, but not the existence of controllers. These tags are also design rules, and their stability becomes important. We added a set of variables in the ACNs, ending with “_DesignRules” to model them.

Stability and Modularity Assessment After ACN modeling, we compute the stability and modularity properties from the 16 ACNs. 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. Our tools compute their pair-wise dependence relation and generate their DSM models. We cluster each ACN using design rule hierarchy (as shown in Figure 1) to compute their *Independence Level*.

We also cluster them into environment/design rule/hidden modules (as shown in Figure 3) to calculate their stability properties, the *Concern Scopes* and *Volatility*. The *Concern Overlap* metric and *Change Impact* metric can be computed based on the variable sets of the ACNs and their derived pair-wise dependence relations.

5.3 Evaluation Results

The assessment results concisely reveal which alternative is better under which circumstances. We first present the assessment results in terms of stability and modularity, and then compare our results with that of the source code analysis conducted by Figueiredo et al [12]. The goal is to check their consistency with the implementation assessment and if our metrics can provide new insights.

Design Stability Assessment Table 1 summarizes the *Concern Scope* measurement. The smaller the number, the lesser the concern or feature is diffused and its impact on stability is smaller. The table clearly shows that AO design is superior in terms of accommodating additions of optional features (the last 4 rows of the table). For the alternative features, the AO version is slightly better, but it does not have significant advantages (in the middle of the table). For the mandatory features, the AO design is generally harmful (the first set of rows) because all the concerns involve more dimensions in the AO version for all the releases.

The left part of Table 2 summarizes the *Design Volatility* measurement. The larger the number the higher the volatility and, therefore, the design is less stable. The table shows that AO designs always appear to be more unstable than OO designs. If we look into the *DecisionVolatility* values (not shown due to space restrictions), we can identify the variable that contribute most to design volatility. For example, the *DecisionVolatility* value of the `AlbumData_Interface` has the highest value in AO release 7- 99, showing that it is both highly crosscutting and subject to change. Its value increases to 137 in OO release 8, showing that one-third of the vulnerability of the system is caused by this dimension. This result is confirmed by domain experts.

The middle part of Table 2 summarizes the *Change Impact* measurement. The table shows that in all 8 releases, the AO designs added more or equal number of variables than the OO designs, and deleted the same number of variables as OO designs. With the exception of version 5 and 7, AO designs requires less or equal changes than OO designs, which means that the AO designs, in general, better satisfy the open-closed principle.

The right part of Table 2 shows the *Concern Overlapping* measurement for two pairs of concerns. Using the method introduced in Section 4, we can easily assess concern overlapping between any pair of features. To be comparable with the results presented in [12], we present the overlap-

		R1		R2		R3		R4		R5		R6		R7		R8	
	Concern	OO	AO														
Mandatory	CreateAlbumFeature	9	9	9	10	10	11	10	11	13	15	14	17	14	18	17	22
	DeleteAlbumFeature	7	7	7	8	7	8	7	8	11	13	12	15	12	16	15	20
	ViewAlbumFeature	7	7	7	8	7	11	7	8	11	13	12	15	12	16	15	20
	NewMedia(CreatePhoto)Feature	9	9	9	10	9	9	9	12	13	17	14	19	14	20	17	25
	DeleteMedia>DeletePhoto)Feature	7	7	7	8	7	8	7	10	11	15	12	17	11	18	15	23
	ViewPhotoFeature	9	9	9	10	9	11	9	12	13	17	14	19				
	ExceptionHandlerFeature			9	10	8	11	8	9	11	15	13	15	18	20	24	28
	EditLabelFeature					9	9	9	12	10	14	10	15	13	20	17	25
Alternative	ViewPhotoFeature													11	11	11	11
	PlayMusicFeature													11	9	11	9
	PlayVideoFeature															11	9
	CaptureMediaFeature															12	10
Optional	CountandSortFeature					3	1	3	1	4	1	4	1	4	1	4	1
	FavouriteFeature							3	1	4	1	4	1	4	1	4	1
	CopyMediaFeature									4	2	5	2	7	2	11	4
	SMSFeature											6	3	6	3	6	3

Table 1. Concern Scope

	Design Volatility				Change Impact						Concern Overlap			
	Volatility		Volatility Increment		Additions		Removals		Change		Controller Vs Labeling		Sorting Vs Labeling	
	OO	AO	OO	AO	OO	AO	OO	AO	OO	AO	OO	AO	OO	AO
R1	63	63												
R2	82	100	19	37	3	4	0	0	7	7				
R3	95	113	13	13	6	6	0	0	8	8	2	4	3	1
R4	99	117	4	4	2	2	0	0	3	0	2	4	3	1
R5	141	164	42	47	8	9	0	0	3	4	3	7	4	1
R6	166	191	25	27	6	7	0	0	3	0	3	8	4	1
R7	182	215	16	24	12	14	1	1	25	26	5	10	4	1
R8	252	305	70	90	16	18	0	0	8	6	9	17	4	1

Table 2. Stability Results

	Independence Level		Difference
	OO	AO	
R1	0.31	0.31	
R2	0.28	0.27	-0.01
R3	0.24	0.31	0.07
R4	0.23	0.34	0.11
R5	0.27	0.36	0.09
R6	0.30	0.39	0.09
R7	0.37	0.44	0.07
R8	0.43	0.47	0.04

Table 3. Design Independence

ping results for the follow two pairs of features: *Controller* vs. *Labeling* and *Sorting* vs. *Labeling*. The table shows clearly that for the first pair, the AO designs caused more overlaps; and for the second pair, the AO designs caused much less overlapping.

Design Modularity Assessment. Instead of using traditional coupling, cohesion and size metrics to measure modularity, we use *Independence Level* (IL) to directly measure the ability of a design to generate option values. Note that this metric is different from Baldwin and Clark’s NOV equation in that the latter measures the option values, but not the ability to produce options. Table 3 summarizes the results. The higher the percentage is, the higher option value can be generated. The table shows the following results: in Release 2, the AO design’s option-generation ability is lower than that of OO design. From Release 3 to Release 6, the IL values of the AO designs are much higher than that of OO designs. The difference is maximized in Release 4. The IL values of OO Release 7 and 8 increase faster and the differences between OO and AO designer are getting smaller. If we look at each columns of the *Independence Level*, we can see that the IL value of OO designs

kept dropping until Release 4. The IL value of the AO design dropped from the first release to the second release, and then kept increasing.

The conclusions include: (1) the AO method is particularly unsuitable for the design of the *ErrorHandling* feature; (2) the AO method is much better in terms of adding optional feature (from Release 3 to 6); (3) the option-generating ability of AO designs for alternative features (Release 7 and 8) is comparable to that of OO designs. The reason is that to add alternative features, the AO designs generate a lot more couplings that diminish the option value aspects created, especially in Release 8. This not the case for Release 3 to 6, where OO and AO generate similar number of couplings but the aspects provides more independent modules. This effect can be seen clearly from the increments of the *Volatility* value in Table 2.

5.4 Comparison against Code Analysis

The prior work of Figueiredo et al. [12] analyzed similar properties from MobileMedia source code. We now compare our architecture level analysis results with their source

code analysis outcomes.

We compare *Change Impact Metric* with their change impact analysis shown in Table 3 [12]. Due to the difference in terms of modeling granularity, comparing the concrete numbers is not meaningful. For example, our *Change Impact* table shows that both OO and AO versions will remove exactly same number of dimensions, which are not similar if lines of code (LOC) and operations are counted. However, if we compare our table with their component-level change impact analysis results, we obtain very similar observations: the AO versions add more dimensions in all releases; the number of removals in their table only differ by 1 between AO and OO. Their conclusion is that the AO versions better follow the open-close principle in general, and we reached the same conclusion.

Our *Concern Scope* and *Concern Overlap* metrics are comparable to their analysis on separation of concerns, and reached highly consistent conclusion with only one exception. In their paper (Section 5.1), they concluded that AO design is better in terms of implementing the `ErrorHandling` concern, which contradicts our conclusion. The differences are granted to the fact that they reach the conclusion by counting LOC. We have confirmed from the authors that AO design performs worse at the component level. Our stability metric set generally concludes that AO design is worse for the `ErrorHandling` (mandatory) concern, which is consistent with their general conclusion about adding mandatory features. The concern overlapping results are consistent with their results.

Our *Volatility* results can be compared with their coupling, cohesion, and complexity assessment. Their results showed that the AO version shows higher level of coupling and lower-level of cohesion in all the releases. While our result showed that the AO versions are more volatile in all releases, which is consistent with their conclusion.

Our assessment provides additional insights though. The *Independence Level* measurements concisely and precisely show under what circumstances the AO design can perform better, indicated by higher IL values. The traditional metrics did not lead to this conclusion directly. Their concern diffusion analysis, which is highly consistent with our concern scope analysis, hinted at this conclusion. But concern diffusion/scope metrics only show how the concerns are spread but not how the concerns are separated from all other concerns. Combining the volatility and independence level results, we can see how the increased coupling diminishes the ability of generating options.

6. Discussions and Future Work

This section discusses the lessons learned and limitations observed while applying our assessment approach to the MobileMedia design releases. We also discuss performance

issues related to the application of our ACN-based assessment proposal.

Lessons Learned and Threats to Validity The starting points of our modeling and assessment approaches are UML component diagrams. The MobileMedia component diagrams used in our study were refined and maintained after implementation. The validity of our evaluation is based on the premise that the MobileMedia component diagrams faithfully reflect the source code structures, so that we can evaluate our results against the detailed source code analysis. The effectiveness of the presented approach thus relies on the accuracy of the UML models. It is possible that UML models do not exist at the beginning or do not conform to the source code structure. The results of our study imply that it is worthwhile to recover and maintain such high-level models, either before or after implementation. Our assessment approach can be applied at either points, helping designers to understand system modularity and stability, predictively or retrospectively, in a concise and scalable way.

The effectiveness of our approach also depends on the designers' ability to identify and express environmental conditions, design rules, and their impact on other design decisions. Two conditions are necessary: the sufficient understanding of the domain knowledge and the generation of logical models. In the presented case study, the ACN construction is only semi-automated. Our future work seeks to automatically derive environmental variables from requirements documents, and formalize and automate the constraints and design rule generation of each design structural patterns. Our observation is that the combination of a component diagram, requirements and patterns may generate sufficiently faithful design models.

It is possible that the change impact metric does not faithfully reflect the effort needed to accomplish the change. For example, comparing the AO vs. OO design of adding `ErrorHandling` in release 2, our design-level analysis showed that the concern spread to more dimensions in the AO version than in the OO version. But the source code analysis showed that the AO implementation involved less lines of code (LOC). This is an inevitable discrepancy caused by the granularity difference between design and implementation, and it is not surprising that source code analysis generates more detailed results.

Some of the ACN-based metrics presented in this paper are simplified to ease their utility, and can be extended if needed. For example, the *Volatility* metrics only count the number of environment variables that impact design decisions, but do not model how likely these environmental conditions will change. It is possible that a design variable suffers impact from many environmental conditions, but none of them will change. As a result, the variable may have a very high volatility value but its volatility can be very low in reality. The volatility metrics can be extended with

the probability to change for each environmental condition. The extended metrics can then be used to conduct sensitivity analysis to assess design stability under uncertainty.

The *Independence Level* (IL) metric is also created with ease-of-use in mind. It is possible that some of the independent modules identified by our design rule hierarchy algorithm contain large number of variables that are tangled together. In this case, even if the IL value is high, the actual level of independence may not be. We plan to extend the metric to take into account the size of each independent module. Baldwin and Clark's NOV model takes into account both the size of each module and the number of modules. The independent modules, an output from our design rule hierarchy algorithm, can be used as part of the input for the NOV model. Our future work also includes the exploration of practical methods to estimate the key parameter in the NOV model, the module technical potential.

Finally, the MobileMedia case study falls into the category of a "retrospective case study". Hence, we are planning to conduct a pro-active case study involving on-going real software projects to assess the predictive ability of these modularity and stability metrics.

Performance Issues We report on the performance of each stage of our case study: the transformation of component diagrams to ACNs, the complementary ACN construction, and the result generation. In the first stage, we developed a small program to translate a component diagram into an ACN. The transformation is instantaneous. The third stage is also not time-consuming using existing ACN processing tools, such as Simon [7]. An ACN model can be processed very quickly, taking about 2 minutes for the biggest models. The second stage involves knowledge-to-ACN translation and is the most time-consuming part of our case study.

Three of the co-authors are familiar with ACN modeling but not the domain knowledge, and the other two authors have sufficient domain knowledge but not ACN modeling. One graduate student at Drexel obtained the initial understanding of MobileMedia from a previous case study [12], and the component diagrams provided by the two co-authors. The graduate student built initial ACN/DSM models based on his understanding. The initial model raised a number of ambiguousness and discrepancies. The student then looked into the source code, exchanged a number of emails with the two domain experts, and held three teleconferences to clarify misunderstandings, and to confirm the correctness of environmental and design rule constraints.

Identifying and modeling which components in the component diagram are impacted by which environmental parameters or design rules took some time too. Complementing the ACN with environment parameters and implicit design rules took about 5 hours for the biggest ACN. The student needed to read the component diagrams very carefully.

The positive experience is that only the direct relations between environmental parameters and components need to be specified. For example, the `ViewPhotoFeature` should impact the `PhotoListScreen` component, which is straightforward though it is time-consuming to record such relations. All the indirect dependencies are identified by the constraint network automatically.

7. Related Works

Several metrics have been proposed in the past years to assess software design modularity and stability [5, 4, 11, 18]. One of the most referenced suites of metrics is the one proposed by Chidamber and Kemerer [11]. Their suite includes metrics for quantifying modularity-related attributes of OO design, such as coupling, cohesion and interface size. Briand et al presents an extensive survey and theoretical validation of coupling and cohesion OO metrics [5, 4]. Recently, Sarkar et al [22] used software metrics to measure software modularity of systems. Their work assesses the benefits of using metrics to guide refactoring of legacy systems in order to make them more modular. Martin [18] defined a metric for quantifying design instability based on the combination of two coupling metrics, named afferent and efferent coupling. Our work differs from all these studies in that we conduct our study at high-level design, while they target source code or detailed design. In addition, these measurement approaches do not consider the influence of environmental variable as our work does. Similarly, metric suites for AO design have recently been developed [6, 10, 29]. Most of these metrics are extensions of OO metrics. As a consequence, they also suffer from the aforementioned limitations of existing OO metrics. Besides, most of the aspect-oriented metrics are defined upon AspectJ abstractions. Our work defines metrics applicable to any modularization technique.

Sullivan et al [24] and Lopes et al [17] have previously assessed and compared software design modularity from a software economics perspective based on Baldwin and Clark's Net Option Value (NOV) [3] analysis. Real Options Theory has also been used by Bahsoon et al [2] to assess architectural stability. However, instead of using one metric, our paper proposes multiple metrics which allow analysis of modularity and stability, supporting design assessment from different perspectives. Cai et al's work [8] is related to our work in that we both assess software design by using modularity metrics on ACN and DSM models. Our work is different in that we use innovative metrics, such as *Independence Level* and *Volatility*, which measure not only modularity but also design rule stability. Also, a significant difference in our work is that our assessment approach considers environmental variables, which represent features that drive the design. Stability measurement of product line architec-

ture has been conducted by Hoek et al [26] using service utilization metrics. However their work is not concerned with measuring stability of modularization techniques and measures one dimension of architectural improvement.

8. Conclusions

To address the problem that current design stability and modularity measurement are conducted based on source code analysis, the lack of unified design-level representations to uniformly assess different modularization techniques, the lack of environmental considerations in prevailing modularity and stability metrics, and the lack of metrics to assess the ability of a design to generate option values, we contribute an approach to translate a UML component diagram into an ACN and then a DSM model, a set of metrics to concisely measure design stability and modularity based on DR/DSM/ACN modeling, and an feasibility evaluation using eight releases of the MobileMedia product line designed and implemented using AO and OO respectively. We show that our design-level assessment approach generates highly consistent results with the previous detailed source code analysis, in a more concise and scalable way, and provides additional insights.

References

- [1] A. Garcia et al. Modularizing design patterns with aspects: A quantitative study. In *Proc. of the 4th AOSD*, Mar. 2005.
- [2] R. Bahsoon and W. Emmerich. Evaluating architectural stability with real options theory. In *Proc. of the 20th ICSM*, Sept. 2004.
- [3] C. Y. Baldwin and K. B. Clark. *Design Rules, Vol. 1: The Power of Modularity*. MIT Press, 2000.
- [4] L. C. Briand, J. W. Daly, and J. Wuest. A unified framework for cohesion measurement in object-oriented systems. *IEEE Trans. on Soft. Eng.*, 3(1):65–117, 1998.
- [5] L. C. Briand, J. W. Daly, and J. Wuest. A unified framework for coupling measurement in object-oriented systems. *IEEE Trans. on Soft. Eng.*, 25(1):91–121, 1999.
- [6] C. Sant’Anna et al. On the reuse and maintenance of aspect-oriented software: An assessment framework. In *Brazilian Symposium on Software Engineering*, pages 19–34, Oct. 2003.
- [7] Y. Cai. *Modularity in Design: Formal Modeling and Automated Analysis*. PhD thesis, University of Virginia, Aug. 2006.
- [8] Y. Cai, S. Huynh, and T. Xie. A framework and tool supports for testing modularity of software design. In *Proc. of the 22nd IEEE/ACM ASE*, Nov. 2007.
- [9] Y. Cai and K. Sullivan. Modularity analysis of logical design models. In *Proc. of the 21st IEEE/ACM ASE*, Sept. 2006.
- [10] M. Ceccato and P. Tonella. Measuring the effects of software aspectization. In *1st Workshop on Aspect Reverse Engineering*, Nov. 2004.
- [11] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *IEEE Trans. on Soft. Eng.*, 20(6):476–493, 1994.
- [12] E. Figueiredo et al. Evolving software product lines with aspects: An empirical study on design stability. In *Proc. of the 30th ICSE*, May 2008.
- [13] F. Filho, A. Garcia, and C. Rubira. Extracting error handling to aspects: A cookbook. In *Proc. of the 23rd ICSM*, Oct. 2007.
- [14] J. Hannemann and G. Kiczales. Design pattern implementations in Java and AspectJ. In *Proc. of the 17th OOPSLA*, Nov. 2002.
- [15] S. Huynh, Y. Cai, and K. Sethi. Design rule hierarchy and analytical decision model transformation. Technical Report DU-CS-08-04, Drexel University, Sept. 2008.
- [16] K. Sullivan et al. Information hiding interfaces for aspect-oriented design. In *Proc. of the 10th FSE*, pages 166–175, Sept. 2005.
- [17] C. V. Lopes and S. K. Bajracharya. An analysis of modularity in aspect-oriented design. In *Proc. of the 4th AOSD*, pages 15–26, Mar. 2005.
- [18] R. Martin. Stability. *C++ Report*, 9(2):54–60, 1997.
- [19] Object Management Group. Unified modeling language: Superstructure. <http://www.omg.org/technology/documents/formal/uml.htm>, Feb. 2007.
- [20] P. Greenwood et al. On the impact of aspectual decompositions on design stability: An empirical study. In *Proc. of the 21st ECOOP*, 2007.
- [21] R. Coelho et al. Assessing the impact of aspects on exception flows: An exploratory study. In *Proc of the 22nd ECOOP*, July 2008.
- [22] S. Sarkar, G. M. Rama, and A. C. Kak. Api-based and information-theoretic metrics for measuring the quality of software modularization. *IEEE Trans. on Soft. Eng.*, 33(1):14–32, 2007.
- [23] D. V. Steward. The design structure system: A method for managing the design of complex systems. *IEEE Trans. on Eng. Management*, 28(3):71–84, 1981.
- [24] 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*, pages 99–108, Sept. 2001.
- [25] K. Sullivan, L. Gu, and Y. Cai. Non-modularity in aspect-oriented languages: Integration as a crosscutting concern for aspectj. In *Proceedings of the 1st international conference on Aspect-Oriented Software Development (AOSD 2002)*, pages 19–26, Sept. 2002.
- [26] A. van der Hoek, E. Dincel, and N. Medvidovic. Using service utilization metrics to assess the structure of product line architectures. In *9th International Software Metrics Symposium*, Sept. 2003.
- [27] S. Yau and J. S. Collofello. Design stability measures for software maintenance. *IEEE Trans. Softw. Eng.*, 11(9):849–856, 1985.
- [28] T. Young. Using aspectj to build a software product line for mobile devices. Master’s thesis, University of British Columbia, Aug. 2005.
- [29] J. Zhao. Measuring coupling in aspect-oriented systems. In *10th International Software Metrics Symposium*, Sept. 2004.