Variations on Using Propagation Cost to Measure Architecture Modifiability Properties

Robert L. Nord[1], Ipek Ozkaya[1], Raghvinder S. Sangwan[1,2], Julien Delange[1], Marco González[3], Philippe Kruchten[3]

Software Engineering Institute[1]
Carnegie Mellon University
Pittsburgh PA, USA
{rn, ozkaya, jdelange}@sei.cmu.edu

Pennsylvania State University[2]
Malvern, PA, USA
rsangwan@psu.edu

Electrical & Computer Engineering[3]
University of British Columbia
Vancouver, Canada
{marcog, pbk}@ece.ubc.ca

***Abstract*—Tools available for measuring the modifiability or impact of change of a system through its architecture typically use structural metrics. These metrics take into account dependencies among the different elements of a system. However, they fail to capture the semantics of an architectural transformation necessary to control the complexity and cost of making changes. To highlight such limitations, this paper presents a study where we applied a representative structural metric, called 'propagation cost', to archetypical architectural transformations known to affect system modifiability such as rearchitecting a tightly coupled system to a layered pattern. We observe that in its original form the propagation cost metric does not provide consistent indications of architecture health. Enhancing this metric based on the semantics of the architectural pattern and tactics used in the transformation show improvements. Our results demonstrate that these enhancements detect modifiability properties that are not detectable by the propagation cost metric.**

Keywords—software architecture; modifiability; propagation cost; stability; change propagation; dependency analysis

I. INTRODUCTION

Modifiability is the quality attribute of an architecture that relates to the cost of change and refers to the ease with which a software system can accommodate change [1]. Techniques that provide guidance on the extent of the modifiability or extensibility qualities of a system often measure code-level metrics such as coupling, cohesion, cycles and code clones. Preventing an architectural change that compromises modifiability, however, is not simply a matter of controlling the quality of a system through the use of these code-level metrics [2]. A recent exploratory study by Bertan *et al.* revealed that many code-related issues detected were not correlated with architectural problems, and many issues that were not detected had architectural implications [3].

More recently, structural metrics based on the use of dependency structure matrices (DSM), also called design structure matrices, have been studied to assist with architecture-level analysis, such as the value of modular designs [4][5][6]. Stability and propagation cost metrics [7] extracted from a DSM view of the architecture indicate the likelihood of change propagation and, consequently, its impact on future maintenance costs and ease of making modifications. Several studies highlight the need to analyze dependencies in detecting design violations such as those in modifiability [8] and provide advice on using other dependency sources to augment the information in a dependency matrix [6]. Increasingly such forms of analysis are being supported by commercial tools such as Sonar, Lattix and Structure 101 [9].

In this paper, we study the DSM-based *Propagation Cost* metric (PC) and we explore the following research questions:

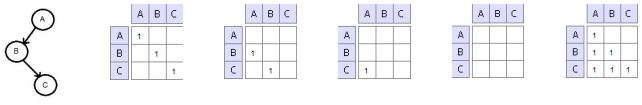
1. Does PC measure modifiability properties to understand the impact of making a change to the architecture?
2. What are the limitations of PC, and how can it be enhanced and calibrated to provide practical guidance in understanding the impact of making a change to the architecture of a given system?

For this purpose, we use a commonly known transformation to improve modifiability: *strictly layered*, where a system is organized into layers with a higher-level layer only allowed to use an adjacent layer immediately below. We apply PC to gauge the state of the system before and after this architectural transformation, thereby highlighting its limitations. We then propose variations of this metric by adding dependency strengths to the DSM view of the architecture. We enhance PC to take into account the information about dependency strength and how it propagates along dependency paths.

While dependency strengths have been used in metrics on code and module structure, they treat dependencies alike. Our objective is to see if improvements to the metrics, using dependency strengths representative of the semantics inherent in the transformation, provide better insights that are in line with the modifiability properties of an architectural design while preserving their usefulness by ensuring such metric improvements can be supported by tools.

II. STRUCTURAL METRICS

Consider a dependency graph showing elements of a system and their dependencies in Fig. 1(a). This dependency graph can be represented as a DSM, shown in Fig. 1(c), where the rows and columns are the elements from the graph and the cells are the dependencies among these elements.



(a) (b) (c) (d) (e) (f)

Fig. 1. (a) Dependency Graph (b) Identity DSM D^0, (c) direct dependency DSM D^1, (d) DSM D^2, (e) DSM D^3, and (f) visibility matrix V

The PC metric characterizes "the degree to which a change in a single element causes a potential change to other elements

1063-6773/13 $26.00 © 2013 IEEE
DOI 10.1109/ICSM.2013.57

in the system, directly or indirectly [7].” To determine the impact of elements in a system, a visibility matrix (V) is computed as follows:

$$V = \sum_{i=0}^{n} D^i \quad (1)$$

Matrices D and V are shown in Fig. 1(b) – (f) for the system shown in Fig. 1(a) where D^0 represents dependencies of elements on themselves, D^1 represents all direct dependencies, and D^p represents all indirect dependences of length p where $2 \leq p \leq n$ and n is the number of elements in a system.

PC, therefore, represents the density (or the proportion of cells marked 1) and is computed using V as follows:

$$PC = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} V_{ij} \quad (2)$$

In the case of Fig. 1, PC is 67% implying a change to an element might affect 67% of the system. This type of analysis forms the basis for the structural metric called stability that tools such as Lattix use to assist with software architecture management [5]. In the rest of the paper, we will focus on PC and use it as a representative structural metric.

III. Architecture Dependency Analysis

We apply PC for analyzing the scenario shown in Fig. 2 where the architecture of a system is transformed using a strictly layered pattern. The *strictly layered pattern* is a division of software into units called layers. The layers are related to each other by the strictly ordered relation, *allowed-to-use*. Transforming a design to be strictly layered consists of removing connections that bypass one or several layers.

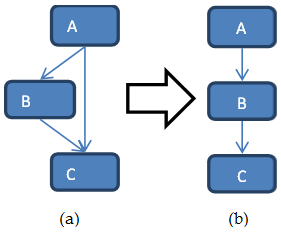


Fig. 2. System (a) before, and (b) after applying strictly layered pattern

In the example in Fig. 2, the connection from A to C bypasses element B. Applying the strictly layered transformation consists of identifying B as the intermediary between A and C and removing the dependency between A and C so that connection from A to C has to pass through B. One would, therefore, expect the architecture of the system to have changed with respect to the modifiability quality attribute depending on the role chosen for B in weakening the dependencies. The result of applying PC (modified to omit the diagonal or identity matrix D^0), however, does not show this. PC is 33% for both before and after the transformation.

To adequately reflect the semantics of the architecture transformation we introduce variations to PC and discuss next.

IV. Variations in Calculating PC

We improve the shortcomings associated with the calculation of PC by focusing on the strength of the dependencies and how they propagate along paths in the dependency graph in series and in parallel.

Consider the dependency graph in Fig. 3.

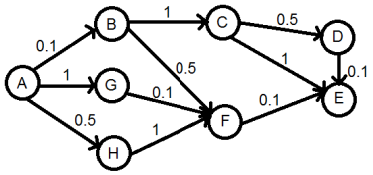


Fig. 3. Dependency graph showing the strength of dependencies

We use values between 0 and 1 for the strength based on measures taken to shield a dependent element from the ripple effect of change made to an element it depends on.

Suppose we have a path of length $n \geq 2$ from element X to Z that goes through element Y such that the path from X to Y is of length $n - 1$ and from Y to Z is of length 1. Then the change propagation cost of dependencies in series, $\text{Cost}(X \rightarrow Z)_n$, is the minimum of the PC of path from X to Y, $\text{Cost}(X \rightarrow Y)_{n-1}$, and PC of path from Y to Z, $\text{Cost}(Y \rightarrow Z)_1$:

$$\text{Cost}(X \rightarrow Z)_n = \min(\text{Cost}(X \rightarrow Y)_{n-1}, \text{Cost}(Y \rightarrow Z)_1) \quad (3)$$

For a path of unit length the change propagation cost is the strength of the dependency associated with that path. Using this convention, the change propagation cost for the path from element A to element D in Fig. 3 would be 0.1. In other words, the only change that propagates from element D to element A is one that is able to ripple through tactics employed along the way in element B to prevent a ripple effect of such a change.

Consider p paths of length n ($n \geq 2$) from X to Z. We use three variants to calculate the change propagation in parallel:

SUM: The SUM of parallel paths from a component X to component Z is computed as follows:

$$\text{Cost}_{\text{SUM}}(X \rightarrow Z) = \sum_{i=1}^{p} \text{Cost}\,(X \rightarrow Z)_n \quad (4)$$

$\text{Cost}\,(X \rightarrow Z)_n$ for each path is computed as in equation (3). Using this convention, the change propagation cost from element A to F is 0.7. What this means is the changes to F are equally likely to propagate along each path from F back to A (A → B → F, A → G → F, and A → H → F) and, therefore, the impact of change on A is the aggregate of change propagation along each path (0.1 + 0.1 + 0.5).

AVG: We SUM the path as discussed above, but divide the result by the number of paths.

$$\text{Cost}_{\text{AVG}}(X \rightarrow Z) = \frac{\text{Cost}_{\text{SUM}}(X \rightarrow Z)}{p} \quad (5)$$

Using this convention, the change propagation cost from element A to F is 0.23. This means that the changes to F are, on average, only likely to propagate along one of the paths from F back to A (A → B → F, A → G → F, and A → H → F). Therefore, the impact of change on A is the average of change propagation along each path ([0.1 + 0.1 + 0.5] / 3).

MAX: We take the value of the most costly path from component X to component Z. Considering we have *p* paths between X and Z, the following is used to compute the cost:

$$\text{Cost}_{\text{MAX}}(X \rightarrow Z) = \max(\text{Cost}_1(X \rightarrow Z)_n, .., \text{Cost}_p(X \rightarrow Z)_n) \quad (6)$$

$\text{Cost}_p(X \rightarrow Z)_n$ is the cost of a given path *p* of length *n* computed using equation (3). Using this convention, the change propagation cost from A to F is 0.5. This means the changes to F are more likely to propagate along a path from F back to A that makes the least effort to prevent such changes from rippling through. Therefore, the impact of change on A is the maximum of 0.1, 0.1 and 0.5, the change propagation along each path.

V. USING VARIATIONS ON STRICT LAYERING PATTERN

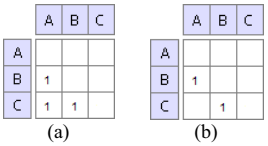
To improve the proposed change propagation cost formulas we will examine how architectural patterns support the modifiability of a system using architectural tactics [10] [11][12]. Modifiability tactics (such as *encapsulate*, *use an intermediary* and *restrict dependencies* that reduce coupling, *semantic coherence* that increases cohesion, and *split module* that reduces the size of a module) are architectural design decisions that control the time and cost of making changes.

The initial and expected strengths of the dependencies and the role of element B in Fig. 2 determine whether the transformation improves or deteriorates the design. We use three values for the strength – small (0.1), medium (0.5) and large (1). A small valued dependency indicates a use of one or more architectural tactics to prevent the ripple effect of a change, whereas a large valued dependency indicates absence of any architectural tactic that could prevent the ripple effect. A medium valued dependency indicates use of some tactics, but they are not sufficient in preventing ripple effects.

We demonstrate three scenarios:

1. Scenario 1, a baseline scenario where all dependencies have equal strength since the role of B is not clear
2. Scenario 2 where B acts as an interface and uses modifiability tactics such as *encapsulate*, *use an intermediary*, and *restrict dependencies* to reduce coupling
3. Scenario 3 where B acts as a conduit and uses modifiability tactics such as *use an intermediary* and *restrict dependencies* to reduce coupling

Scenario1 simulates the original PC outlined in section II and gives us a reference point to compare the variations. Fig. 4 shows DSMs of the system before and after applying the strictly layered pattern and change propagation cost.



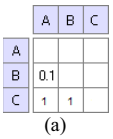
Formula	Cost Before	Cost After
PC	33%	33%
Sum	44%	33%
Avg	33%	33%
Max	33%	33%

(c)

Fig. 4. Scenario 1 DSM (a) before and (b) after applying strictly layered pattern; (c) change propagation cost

The original PC and the formula variations we suggest compute the same result with the exception of *Sum* for the before case. *Sum* takes into account the changes that may propagate from both of the parallel paths which may be cumulative. Therefore, *Sum* shows an improvement since the parallel paths between A and C are broken. Using the original PC of this scenario associates the same strength with each dependency and does not always accurately reflect the semantics of the change in the design.

Fig. 5 shows DSMs before and after applying the strictly layered pattern in scenario 2. In this scenario, A uses the interface B, but also has a direct connection to C. After the transformation, the dependency weights reflect the strictly layered pattern where B acts an intermediary and weakens the dependencies between A and C.



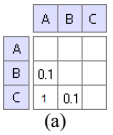
Formula	Cost Before	Cost After
Sum	24%	13%
Avg	18%	13%
Max	23%	13%

(c)

Fig. 5. Scenario 2 DSM (a) before and (b) after applying strictly layered pattern; (c) change propagation cost

The transformation uses the tactics *use an intermediary* and *restrict dependencies* to break the dependency between A and C. Furthermore, the tactic *encapsulate* is applied so B doesn't just pass on all information from C to A. The semantics of the transformations indicate that the change propagation of the structure has improved since the strong dependency between A and C is broken. All of the variations indicate an improvement.

Fig. 6 shows DSMs before and after applying the strictly layered pattern in scenario 3. In this scenario, B is loosely coupled with A and C and does not act as an interface between these two components. After the transformation, B has to know more about C to act as an intermediary for A. Thus, removing the direct connection between A and C increases the dependency strengths between A and B and B and C. Indeed, all information passing between A and C (that are strongly dependent) would go through B.



Formula	Cost Before	Cost After
Sum	14%	33%
Avg	8%	33%
Max	13%	33%

(c)

Fig. 6. Scenario 3 DSM (a) before and (b) after applying strictly layered pattern; (c) change propagation cost

The transformation uses the tactics *use an intermediary* and *restrict dependencies*, but without enforcing information hiding imposed by the *encapsulation* tactic. The semantics of the transformations indicate that the structure has deteriorated since previously A and B and B and C were loosely coupled. After the transformation they are strongly coupled since B is acting as a conduit to provide all information from C to A. The variation formulas indicate a change for the worse.

VI. DISCUSSION AND CONCLUSIONS

Our study focused on two research questions. First, we investigated if the existing DSM-based structural metric, PC, reflects the impact of making a change to the architecture of a given system. Our example pattern of architectural transformations shows that in its current form PC does not reflect the semantics of architectural transformations. This

finding is in line with recent studies that look at the code metrics [3], and extend that to structural metrics [13].

Second, we explored different ways of enhancing and calibrating PC to provide practical guidance in understanding the impact of making a change to the architecture of a given system. Our enhancements were motivated by several limitations in the way change propagation cost is calculated:

- *The current propagation cost metric takes into account the presence or absence of a dependency.* We added dependency strength to the model and included formulas for handling change propagation in series and in parallel.
- *It assumes that all components are equally likely to be changed when the system evolves.* Dependency strength and certain formulas for change propagation in parallel (e.g., *Avg*, *Max*) weaken the assumption that all components contribute equally.
- *In case of smaller number of components, the contribution of self-dependencies is too high.* Archetypical transformations have very small matrices. This makes the metric very sensitive to a change when self-dependencies are considered, a phenomenon not observed in large systems (e.g., when the dimension is greater than 12). We omit the diagonal (the identity matrix D^0) in the calculation to reduce the influence of a smaller number of components.
- *The transitive effect of indirect dependencies is too high.* Our use of dependency strength and formula for change propagation in series take into account the muting effect of any tactics used to prevent a change from rippling.

Our initial results hold promise for improving existing structural metrics, such as stability and PC, with common architectural patterns semantics, illustrating the range of design parameters for reducing the size of a module, increasing cohesion, and reducing coupling, and the tactics for manipulating them. We applied the same approach to two other architectural transformations, *module split* where a large element is split into two or more so each has fewer dependencies and *short circuit* where connections are bypassed by direct communication. We computed similar results and in the interest of space reported on the *strictly layered* pattern.

A validity concern for this study is whether the variation formulas will still perform consistently when applied in large systems. In addition, in some of our scenarios *Sum* and *Max* show greater ranges of improvement compared to *Avg*. To investigate whether this range is meaningful and whether any one of the variations among *Sum*, *Avg,* or *Max* perform better consistently, we are applying our variation metrics on an open source health IT project, CONNECT, where we have insights into its architecture of version 3.x through an architecture evaluation study we were asked to conduct [14].

The results of our study present an incremental, but significant improvement over the current state of practice showcasing how existing structural metrics such as PC and stability may yield results that can be misleading. Our approach emphasizes the significance of design decisions (architectural patterns and tactics) in moderating change propagation. While this is not surprising, the ability to measure the influence of these decisions in a way that can inform the architects and instill confidence in the choices they make is significant.

ACKNOWLEDGMENT

We thank Neeraj Sangal and Frank Waldman for their feedback in automating the analysis. DSM figures are drawn using Lattix.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. DM-0000486.

REFERENCES

[1] ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

[2] R. Marinescu and D. Ratiu. Quantifying the Quality of Object-Oriented Design: The Factor-Strategy Model, in Proc 11th Working Conference on Reverse Engineering, Delft University of Technology, The Netherlands, November, 2004, pp. 192-201.

[3] M.I. Bertan, J. Garcia, D. Popescu, A. Garcia, N. Medvidovic, A. von Staa. Are automatically-detected code anomalies relevant to architectural modularity?: an exploratory analysis of evolving systems. AOSD 2012: 167-178

[4] C. V. Lopes and S. Bajracharya. 2005. An Analysis of Modularity in Aspect-Oriented Design. In Proceedings of Aspect-Oriented Software Development. Chicago, IL, March 14-18, 2005. ACM.

[5] C. Hinsman, N. Sangal, J. Stafford. Achieving agility through architecture visibility. in Proc QoSA 2009, pp. 116–129, 2009.

[6] T. Callo Arias, P. van der Spek, & P. Avgeriou, (2011). A practice-driven systematic review of dependency analysis solutions. Empirical Software Engineering 1-43.

[7] A. MacCormack, J. Rusnak, and C. Y. Baldwin. "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code." Management Science 52, no. 7 (July 2006).

[8] S. Wong, Y. Cai, M. Kim, M. Dalton: Detecting software modularity violations. ICSE 2011: 411-420

[9] A. Telea, L. Voinea, H. Sassenburg. "Visual Tools for Software Architecture Understanding: A Stakeholder Perspective," Software, IEEE , vol.27, no.6, pp.46-53, 2010.

[10] F. Bachman, L. Bass, and R. Nord. 2007. Modifiability Tactics. Technical Report CMU/SEI-2007-TR-002 September 2007. Carnegie Mellon Software Engineering Institute. http://www.sei.cmu.edu/library/abstracts/reports/07tr002.cfm

[11] C. Hofmeister, P. Kruchten, R.L. Nord, H. Obbink, A. Ran, P. America. A General Model of Software Architecture Design Derived from Five Industrial Approaches, Software Architecture Section, Journal of Systems and Software, Elsevier, 2007.

[12] N. Harrison, P. Avgeriou, How do Architecture Patterns and Tactics Interact? A Model and Annotation, J. of Systems and Software, Elsevier, vol. 83, no. 10, October 2010, pp. 1735-1758.

[13] J. Brondum and L. Zhu, "Visualising architectural dependencies," Managing Technical Debt (MTD), 2012 Third International Workshop on , vol., no., pp.7,14, 5-5 June 2012.

[14] N. A. Ernst, I. Ozkaya, R. L. Nord, J. Delange, S. Bellomo, I. Gorton, "Understanding the Role of Constraints on Architecturally Significant Requirements" 3rd Int. Workshop on the Twin Peaks of Requirements and Architecture, July 2013.