

Detecting Architecture Instabilities with Concern Traces: An Exploratory Study

Eduardo Figueiredo¹, Ismênia Galvão², Safoora Shakil Khan¹, Alessandro Garcia³,
Claudio Sant'Anna³, Afonso Pimentel⁴, Ana Luisa Medeiros⁵, Lyrene Fernandes⁵, Thais Batista⁵,
Rita Ribeiro⁴, Pim van den Broek², Mehmet Aksit², Steffen Zschaler¹, and Ana Moreira⁴

¹Computing Department, Lancaster University, United Kingdom

²Software Engineering Group, University of Twente, The Netherlands

³Opus Research Group, Informatics Department, PUC-Rio, Brazil

⁴Informatics Department, Universidade Nova de Lisboa, Portugal

⁵Computer Science Department, University of Rio Grande do Norte, Brazil

{figueire, shakikh, z.schaler}@lancaster.ac.uk, {galvai, broekpm, aksit}@ewi.utwente.nl, {afgarcia, claudios}@inf.puc-rio.br, {afonso, rita, amm}@di.fct.unl.pt, {analuisafdm, lyrene, thais}@ufrnet.br

Abstract

Sustaining architecture stability in incremental software development is an important aim for software engineers. Traceability mechanisms can be used to assess and predict architecture stability based on recorded information of early software artefacts. However, there is little empirical knowledge on whether traceability of stakeholders' concerns can assist the identification of architecture instabilities. This paper reports on a first exploratory study that analyses the effectiveness of concern traces for architecture stability assessment. We investigate to what extent properties of concern traces, such as their shapes, are correlated with architectural instabilities. Our analysis is based on eight releases implementing two versions of a software product line for handling mobile media.

1. Introduction

Software architecture documents the key design decisions to satisfy stakeholders' concerns. A concern is any critical or important consideration to stakeholders involved in software development [14]. Concerns are realised first in requirements (e.g., use cases), but they can also emerge later and affect multiple artefacts (e.g., component models) [14]. With the increasing volatility of stakeholders' concerns, prolonging architecture stability is an overarching aim to software engineers [10].

Traceability mechanisms [9, 13] provide basic means to anticipate and analyse software architecture stability. Currently, architecture-level traceability is

usually implemented in the form of component-driven traces [5, 13, 15]. That is, they only record dependencies between an architectural component and specific elements of other artefacts, such as use cases.

Some authors have recently claimed that concern traces [4, 14] improve predictability of software instabilities [6]. A concern usually spans a number of scattered architectural decisions and, possibly, is associated with multiple requirements [3, 4, 14]. For instance, concern traces allow describing which elements in architectural artefacts were influenced by a stakeholder's concern. However, all studies analysing the value of concern traces focused on implementation artefacts [4, 7] rather than on early software development artefacts.

This paper presents a first exploratory evaluation on the effectiveness of concern traceability for assessing architecture stability. More specifically, we address the following research questions:

- RQ1: “Can traces of requirements-level concerns help to anticipate instabilities?”
- RQ2: “Are certain properties of concern traces correlated with architecture instabilities?”
- RQ3: “Which component instabilities can be predicted with concern traces?”

To answer these questions, we analysed component instabilities in 8 releases and 2 versions of a product-line architecture, called MobileMedia (Section 2). We also used trace links of the MobileMedia concerns to assess their ability to predict instabilities. Section 3 analyses to what extent properties of concern traces support the identification of instability sources. Section 4 gives the concluding remarks.

2. Study Settings

This section describes the product-line architecture (PLA) and empirical procedures used in our exploratory study (Section 2.1). The PLA selection was followed by the choice of concerns to be traced (Section 2.2). Section 2.3 describes underline properties of concern traces, such as concern shapes.

2.1 The Mobile Media Product Line

Our exploratory study is based on an evolving PLA, called MobileMedia (MM) [8]. It is a software product line for applications that handle photo, video, and music data on mobile devices, such as cellular phones. Eight MM releases, available in both Java and AspectJ [12], were generated over the last two years. Each release evolved from the previous one by addressing change scenarios, which range from modifications in non-functional requirements to changes in varying or mandatory features. The detailed description of the change scenarios can be found elsewhere [8].

Various reasons justify the choice of MobileMedia. First, multiple releases and two architecture versions are available, each of them introducing realistic change scenarios. Second, requirements, architecture, and implementation artefacts are also available for all releases [3, 8]. Third, we were able to recover and trace the key design decisions together with the original developers. Last, the availability of more than one version allowed us to evaluate the usefulness of concern traces in different architectural styles.

2.2 Capturing Architectural Concern Traces

A set of 14 relevant stakeholders' concerns were traced through requirements and architecture artefacts for all MM releases of the object-oriented (OO) and aspect-oriented (AO) architectures. The selected concerns are representative of varying and mandatory features and include Album, Capture, Controller, Copy, Favourites, Label, Media, Music, Photo, SMS, Sorting, Video, ExceptionHandling, and Persistence. Some of the analysed concerns were mostly relevant in architecture artefacts (e.g., Controller). These concerns were selected as they represent the key PLA drivers according to either original or release documentations.

Changes in some of the MM concerns introduce intricate ripple effects with impact at several levels. Figure 1 presents a representative architectural view of the MM Release 6. This figure also tags the partial propagation of six concerns in architectural elements of this release (using ovals). Gray scale indicates how much a component is dedicated to the concern

realisation; from 'dark grey' which means high dedication to 'white' which means little dedication. Both OO and AO architecture versions are mainly based on the MVC pattern [2]. An 'R' mark on the bottom of a component indicates whether this component was added (+R) or changed (~R) in a specific release. For instance, MusicPlayController and MusicAccessor were added in the sixth release (+R6).

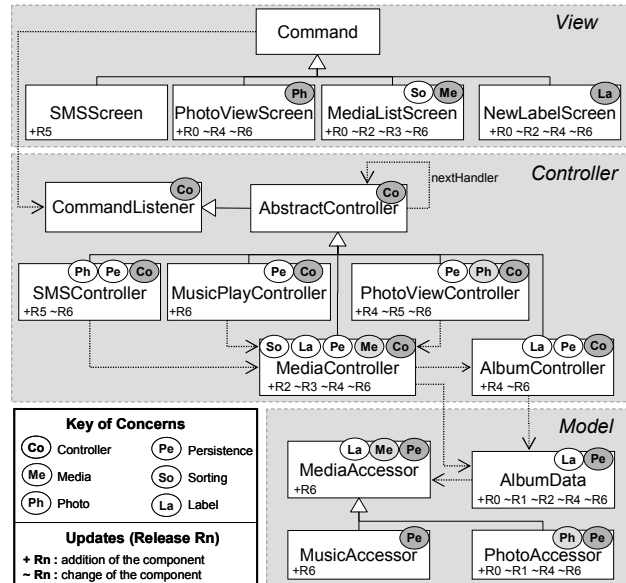


Figure 1. Architectural concern traces in MM R6.

2.3. Classifying Architectural Concern Shapes

The collection of all concern traces enables us to identify five recurring patterns of concern shapes, namely Black Sheep, Octopus, King Snake, Tsunami, and Tree Root, in the MM architecture models. These patterns were also defined and documented in our previous code-level concern analyses [7]. Black Sheep is described as a specialised category of concern that affects only a few scattered architectural elements (e.g., Sorting in Figure 1). Octopus is a concern that is partially well modularised by one or more components (the *octopus body*), but it is also spread across a number of other components (the *octopus tentacles*). Label is an example of Octopus in Figure 1.

The third pattern, King Snake, classifies a concern realises by a large non-cyclic chain of connected components (e.g., Persistence in Figure 1). The first component in the chain is named *head* of the snake and the final connected component (end of the chain) is named *tail*. The Tsunami concern is formed by a core component, named *wave source*, which directly or indirectly connects to other components realising the same concern. Inversely, a Tree Root concern is formed by the *trunk* component that receives incoming

connections from other components (called *feeders*). Figure 1 shows a Tree Root instance formed by components realising the Controller concern. All instances of concern shapes detected in the MM architecture are listed in the project website [1]. Details about patterns of concern shapes can be found at [7].

3. Concern Traces: Results and Discussion

This section describes the key results of our analysis of inter-level traces (Section 3.1), concern shapes (Section 3.2), and design decisions (Section 3.3).

3.1. Inter-Level Concern Traces

Inter-level concern traces record how certain requirements-level concerns manifest in architectural models. Based on such traces, we analyse two kinds of dependencies [11]: *Overlap* and *Intertwine*. An *Overlap* dependency exists when two or more concerns share the same elements of a component. The *Intertwine* dependency, however, occurs when two or more concerns are realised by disjoint elements of the same component. Table 1 shows the measures of *Intertwine* and *Overlap* of MM Releases 4 to 7.

We observed from our analysis that the higher values of intertwining indicate unstable components in both AO and OO architectures. This situation can be better verified in Release 6 when many changes occurred by the introduction of the Music concern.

Table 1. Intertwine (I) and Overlap (O) dependencies.

Components	R4		R5		R6		R7		
	I	O	I	O	I	O	I	O	
MediaController	10	7	10	7	10	7	14	6	*
AlbumController	7	5	7	5	7	5	7	5	
MediaListScreen	6	2	6	2	8	1	8	4	*
SMSController	6	2	6	2	6	3	6	3	*
AlbumListScreen	5	0	5	0	5	1	5	1	
PhotoViewController	4	6	4	9	4	11	4	15	*
AddMediaScreen	3	2	3	4	4	4	4	6	*
NetworkScreen	3	0	3	0	3	0	3	0	
NewLabelScreen	3	1	3	1	3	1	3	1	*
MediaListController	2	1	2	1	2	1	2	1	
PhotoViewScreen	0	2	3	3	3	3	4	3	*
AlbumData	0	22	0	24	0	29	0	24	*
ImageAccessor	0	12	0	13	0	14	0	14	*
PlayMusicScreen					5	1	5	1	
PlayMusicController					6	6	6	6	

* Indicates unstable components (see our website [1]).

We found that Intertwine tends to be a good indicator for confined changes. For instance, AlbumController suffered only few changes when the alternative features were introduced in Releases 6 and 7. From this analysis, we learned that it is important to take into consideration (i) the change intent, (ii) the nature of the involved features (mandatory, optional, alternative), and (iii) other characteristics of concerns (e.g., functional or non-functional requirements).

3.2 Concern Shapes as Instability Indicators

This section assesses the accuracy of concern shapes to predict component-level instabilities. Our initial general assumption was that the more concerns a particular component realises (independently from their shapes), the more unstable this component is likely to be. Then, we used this assumption to identify two sets of components: (i) components that are strong suspects of being unstable and (ii) components that are not suspected of being problematic. The latter set has components realising few concerns. Finally, we contrasted each set with the list of actual unstable components (see our website [1]).

Table 2. Unstable components and respective concern shapes in Release 6.

Architectural Elements	Concern Shapes (OO)					Concern Shapes (AO)				Num. of Shapes		Num. of Changes	
	BS	Oct	KS	Tsu	TR	Oct	KS	Tsu	TR	OO	AO	OO	AO
AlbumController	0	6	4	3	7	5	4	3	6	20	18	1	1
AlbumPhotoData	0	5	6	6	6	5	6	6	6	23	23	4	4
ImageAccessor	0	4	5	5	5	3	5	5	4	19	17	3	3
MediaController	2	7	8	7	10	6	7	7	8	34	28	4	3
PhotoViewContr.	2	7	4	3	9	6	3	3	7	25	19	3	3
<<Complete list of components are available at [1]>>													
AddMediaScreen	0	4	0	2	1	4	0	2	1	7	7	5	5
AlbumListScreen	0	2	2	2	2	2	1	1	2	8	6	2	2
NetworkScreen	0	1	1	1	1	1	0	1	1	4	3	0	0
NewLabelScreen	0	2	2	1	2	2	2	1	2	7	7	3	3
PlayMusicScreen	0	2	1	1	1	1	0	1	1	5	3	0	0
Average	0.4	4	3.3	3.1	4.4	3.5	2.8	3	3.8	15.2	13.1	2.5	2.4

Table 2 illustrates part of our results by showing the top-five components (holding more concerns) and the bottom-five components (holding fewer concerns) in both AO and OO architectures. In addition to the number of concern shapes, Table 2 also presents the number of changes in either architecture options (last two columns). Columns 2-10 specify the number of instances for each concern shape: Black Sheep (BS), Octopus (Oct), King Snake (KS), Tsunami (Tsu), and Tree Root (TR). No Black Sheep instance was found in the AO Release 6 (hidden column).

High Incidence of Concern Shapes Correlates with Component Instabilities. An interesting situation can be recognised in this table: components with higher occurrences of changes are also locus of more concerns exhibiting different shapes (top components). For example, four out of five components with about 20 concern shapes changed at least three times through the MM evolution. On the other hand, three out of five components with no more than 10 concern shapes changed twice at most. This result suggests that our analysis based on the overall number of concern shapes can be used as an intuitive indicator of PLA instability. In other words, a high occurrence of concern shapes in a component is likely to entail design instabilities in later PLA releases.

3.3. Design Decisions and Evolutionary Steps

We also investigate how design decisions can help to detect instabilities. Instead of just looking at loosely related sets of traces, in this evaluation we reason beyond the triggering of evolution activities. The core design decisions that refer to either requirements or architecture issues were recorded and their validity monitored. We found that the collected information about design decisions explains false positives and false negatives obtained in previous analyses.

For instance, the Persistence concern is reflected in the system by means of basic activities over data which can be invoked from multiple points during the software execution, such as include, delete, and update (data). Two decisions deal with persistence in MobileMedia: DataPersistence and DataProvision. The latter crosscuts several use cases, concerns, and architectural components. Both DataPersistence and DataProvision restrict the user's choice to assign an album for every media item. This implies that album names are passed as parameters to several operations and, therefore, justifies the high number of components forming concern shapes for the Album concern. By contrast, the component AlbumController classified as unstable by the analysis based on concern shapes (Section 3.2) is in fact a false positive. This component is created based on the ControllerStrategy design decision in Release 4 and it only changes once in Release 6 due to the renaming of 5 operations. These operations were renamed from Photo to Media, e.g., from newPhotoAlbum to newMediaAlbum.

An example of false negative in the analysis of Section 3.2 is the stability of the AddMediaScreen component. Although this component only requires and provides a couple of operations, it changes in 5 out of 8 releases (in both AO and OO editions). In fact, the cause for this instability is mostly related to the DataProvision decision which evolves in the last three releases. Therefore, we can observe in this case that, although few concern shapes are linked to the component, this fails to predict changes related to a decision concerning memory restrictions.

4. Concluding Remarks

This paper reported an exploratory study on the stability assessment of software architecture based on concern traces. Our assessment methodology includes a complementary set of concern-based analyses, such as inter-level concern traces and concern shapes. Our results for requirements-architecture concern traces show that overlapping dependency is more common in OO than AO PLA (Section 3.1). On the other hand,

intertwine dependency presents a high absolute number and occurs more often in AO PLA.

Based on our analysis of concern shapes (Section 3.2), we verified that many instances of concerns shapes in a component may lead to its instability. Finally, the results obtained by recording the design decision rationale (Section 3.3) complement the other two analyses. For instance, it was able to pinpoint limitations (i.e., false positives and false negatives) of the analysis based on concern shapes.

Acknowledgments. This work is partially supported by the European Commission (grant IST-33710, AMPLE Project) and by two Brazilian Funding Agencies (CAPES and CNPq).

5. References

- [1] <http://www.comp.lancs.ac.uk/~shakilkh/MMData/>
- [2] F. Buschmann *et al.* "Pattern-Oriented Software Architecture: System of Patterns". John Wiley, 1996.
- [3] J. Conejero, E. Figueiredo, A. Garcia, J. Hernandez, and E. Jurado. "Early Crosscutting Metrics as Predictors of Software Instability". Proc. of TOOLS-Europe, 2009.
- [4] B. Dagenais, S. Breu, F. Warr, M. Robillard. "Inferring Structural Patterns for Concern Traceability in Evolving Software". Proc. of ASE, 254-263, 2007.
- [5] A. Dutoit *et al.* "Rationale Management in Software Engineering: Concepts and Techniques". Rationale Management in Software Engineering, 1-48, 2006.
- [6] M. Eaddy *et al.* "Do Crosscutting Concerns Cause Defects?" IEEE TSE, 34(4), pp. 497-515, 2008.
- [7] E. Figueiredo *et al.* "Crosscutting Patterns and Design Stability: An Exploratory Analysis". Proc. of the Int'l Conf. on Program Comprehension (ICPC), 2009.
- [8] E. Figueiredo *et al.* "Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability". In Proc. of Int'l Conf. on Software Engineering (ICSE), 2008.
- [9] O. Gotel and A. Finkelstein. "An Analysis of the Requirements Traceability Problem". Proc. of the Int'l Conf. on Requirements Engineering (RE), 94-101, 1994.
- [10] D. Kelly. "A Study of Design Characteristics in Evolving Software Using Stability as a Criterion". IEEE TSE, 32(5), pp. 315-329, 2006.
- [11] S. Khan *et al.* "On the Impact of Evolving Requirements-Architecture Dependencies: An Exploratory Study". Proc. of CAiSE, 243-257, 2008.
- [12] G. Kiczales, *et al.* "An Overview of AspectJ," Proc. of ECOOP, pp. 327-353, 2001.
- [13] B. Ramesh, M. Jarke. "Toward Reference Models for Requirements Traceability". IEEE Transactions on Software Engineering, 27(1), 58-93, 2001.
- [14] M. Robillard, G. Murphy. "Representing Concerns in Source Code". ACM TOSEM, 16(1), 2007.
- [15] A. Tang, Y. Jin, J. Han. "A Rationale-based Architecture Model for Design Traceability and Reasoning". J. of Syst. and Soft., 80(6), 918-934, 2007.