On The Design of an End-to-End AOSD Testbed for Software Stability

Phil Greenwood[1] Alessandro Garcia[1] Thiago Bartolomei[2]
Sérgio Soares[3] Paulo Borba[4] Awais Rashid[1]

[1]Lancaster University, UK [2]University of Waterloo, Canada [3]Pernambuco State University, Brazil [4]Federal University of Pernambuco, Brazil

{greenwop,garciaa,marash}@comp.lancs.ac.uk, sergio@dsc.upe.br, phmb@cin.ufpe.br, ttonelli@swen.uwaterloo.ca

1. INTRODUCTION

With AOSD techniques gaining attention in academic and industrial settings, it is vital that empirical evaluation is performed to determine their positive and negative impacts on the entire software development process. Assessments of AOSD are currently limited due to two main factors. First, they are currently performed in isolation of individual lifecycle stages, bearing little correlation to preceding and subsequent stages. Second, little attention is given to pivotal maintenance-related attributes, such as software stability. As a result, we have not been able to build a clear understanding of the effects of AOSD when compared to other modularization techniques, especially in realistic constantly-changing development scenarios.

This paper reports initial issues on the design of a testbed to provide end-to-end systematic comparison of AOSD with other mainstream development approaches. The project began in June 2006 and has currently involved eight research institutions. The goal is to allow proponents of AO and non-AO modularization techniques to compare their approaches in a consistent manner at all stages of the software development process. The core of this testbed is presently based on: (i) a common application (benchmark) that involves heterogeneous types of crosscutting and non-crosscutting concerns, and (ii) a suite of metrics for assessing and comparing (but not limited to) software stability phenomena in different non-AO and AO lifecycle artefacts. In particular, the testbed is also being designed to support end-to-end analysis of evolving concern interactions and how they manifest themselves through the development process under observation.

2. DESIGN METHODOLOGY

Startup Benchmark. As stated in the introduction, the purpose of the testbed is to provide a set of benchmarks (and associated metrics) for the end-to-end comparison of AO and non-AO techniques. In order to achieve this, a core benchmark application had to be selected. Certain desirable criteria were established to support the choice of an appropriate application (see Table 1, 1st column). The Health Watcher system [1] was the selected system (Section 3) from ten candidate applications.

Selection of the Target Software Artefacts. We have initially targeted four key software development phases, including: requirements specification, architecture, detailed design, and implementation. The selection of specific software development artefacts to be observed in these phases was shaped by the results of a questionnaire distributed to a set of non-AO and AO research groups. The purpose of this questionnaire was also to determine key quality indicators in each of the software development phases so that the initial measurement suite could provide relevant data. For example, requirement engineering approaches commonly make use of precision and recall metrics so it is important that such measures are included in the testbed.

Determining End-to-End Assessment Dimensions. It was necessary to decide the assessment mechanisms that should be made available for all the development stages. Commonalities between each development phase had to be found so that phase-to-phase evaluation traceability is possible. Universal assessment mechanisms were defined to prevent the phases being assessed in isolation. One of the common themes identified was the issue of concern interaction. Each development phase has some notion of concern interaction either between aspect and base elements or between aspects. This lead to the selection of metrics that support the analysis on how well AO and non-AO approaches handle these interactions in the presence of artefact changes. Hence, desirable and undesirable phenomena could be observed, such as ripple effects, reduction of the number of undesirable interactions, and support for stable interactions (i.e. the number of interactions remained the same given certain changes). In addition, since the testbed is particularly focused on the analysis of software stability, we have also decided to include metrics for assessing change propagation throughout the software lifecycle artefacts. Finally, modularity indicators entail paradigm-agnostic measures that have been historically used (and validated) to assess the quality of artefacts in all software development phases. Examples of such indicators include: coupling, cohesion, separation of concerns, and conciseness [2, 3]. Although traditionally some of these modularity metrics have been applied to code and design artefacts, work has been performed to customise and apply related metrics to other developments phases, such as architecture design [4] and even requirements [5]. The initial metrics suite is presented in Table 1, 3rd column.

Testbed Usage Policy. It is important to point out that all the above design decisions are only a set of initial steps to implement a base testbed which will be built upon. The aim is that the software engineering community should contribute to the initial case studies in any way they see fit. This may include adding additional phases (such as testing), including the analysis of other software artefacts, or performing different case studies to gather more results from existing development phases. The only requirement of anyone using the testbed is that they contribute with concrete results they generate from using the testbed. This will create openness in the community not previously established with the view of a common application/testbed being used in a variety of research projects.

3. ELEMENTS OF THE BENCHMARK

Health Watcher Characteristics. As stated previously at the core of the testbed is the Health Watcher (HW) system, a typical Web-based application that manages health-related complaints. It encompasses a variety of crosscutting concerns (e.g. concurrency, persistence, distribution, exception handling, data management) and non-crosscutting concerns. Also, it does not implement low-level technical concepts that are difficult to understand, for example, commonly-used technologies such as Servlets, JDBC, RMI, GUI and design patterns are employed. Importantly for the testbed, the non-

Criteria		Target Techniques			Examples of Metrics		
Criterion	Purpose		Non-AO	AO	Modularity (Coupling, cohesion, size, Soc)	Change Propagation	Concern Interaction
System generality	Increase comprehensibility	Req.	Use-Case* V-Graph*	Viewpoint-based AORE* AORA* MDSOC* AOV-Graph*	Number of use-cases that extend or enclose another use-case, number of steps in a use-case, number of use-cases that shape one concern	Concerns affected. Compositions affected. Trade-offs affected.	Not applicable – this analysis is already a discrete process of RE approaches.
AO and non-AO implementation availability	Reduce time in creating the testbed, focus on creating benchmarks						
Available Life-cycle documentation	See above.	Arch	UML ACME	AO Templates Aspectual ACME AOSD-Europe Notation*	Fan-in/out, lack of component cohesion, interface complexity, concern diffusion over architectural components.	Architectural components affected, interfaces affected.	Interfaces contributing to concern interactions.
Concern interactions present	Provide interesting analysis when artefacts are evolved.						
Development phase aspects emerge (e.g. aspects should appear/disappear at different phases)	Provide better inter-phase analysis.	Design	UML Theme	aSideML	Coupling between design components, lack of cohesion over operations, vocabulary size, concern diffusion of design components	Changed design modules, changed operations	Components contributing to concern interactions
Aspect types present (domain specific vs. general)	Prevents bias to one particular set of aspects.	Impl.	Java*	AspectJ* CaesarJ* Hyper/J	Coupling between components*, lack of cohesion over operations*, vocabulary size*, concern diffusion over lines of code*.	Changed components*, changed operations*, changed relationships*	Components contributing to concern interactions*

Table 1. Column 1 shows the criteria used to select the base application. Column 2 lists the various AO and non-AO approaches applied to the testbed. Column 3 provides a selection of metrics to be applied the testbed. Elements marked with * have already been implemented/measured.

AO and AO HW designs were both driven by reusability, maintainability and stability requirements. This will help testbed users to ensure that any comparisons between AO and non-AO techniques are fair.

A Pilot Case Study. A pilot implementation-level case study has been carried out to allow us to define initial improvements for the testbed. As it can be seen in Table 1 (last row of 2nd column), one OO and two (rising to three) AO implementation versions of HW were created and compared. We have applied the metrics suite (Section 2) to all the three versions. A tally of nine heterogeneous types of changes in all HW code artefacts and respective measurements were also performed. It allowed us to carry out an initial stability assessment of the Java, AspectJ, and CeasarJ implementations. This analysis also included the observation on how concern interactions evolved in all the three HW versions.

Further Case Studies. Together with the other research groups involved, we are planning to perform similar case studies in the other development phases. Whereby using a range of AO and non-AO techniques (listed in Table 1, 2nd column). Although the selected metrics defined for each phase are not identical (Table 1, 3rd column), they have clear relationships allowing data to be correlated with preceding and subsequent phases. They will support the identification of differences in the manifestations of concerns within each phase, and how the interactions of these concerns evolve. A related set of changes will be then applied to the requirements, architecture and design phases to evolve these artefacts in a similar manner as in the implementation case study.

4. TESTBED EVOLUTION

The development of the first pilot case study raised certain issues that would restrict the testbed effectiveness. The first problem that arose involved the diversity of changes applied at the implementation phase. Although these changes selected had the desired impact at the code level, similar effects could not observed at earlier phases, particularly at the architecture design phase. Only changes that relate to error handling (a typical architecturally-relevant concern) allowed us to analyse interesting stability issues in architectural artefacts. As a result, further changes have to be generated that will have a greater impact on the architectural phase.

Some of the metrics can generate large amounts of data and require a great deal of analysis, for example, analysing concern interactions has potentially many permutations and combinations. As such, it is sometimes difficult to focus such analysis on relevant or interesting elements. In these cases, data collected from other metrics can be used to direct such analysis. For example, the results gathered from the modularity metrics were initially used to pick potentially interesting combinations of concerns (i.e. concerns that exhibit large variation in there modularity metric results) for concern interaction analysis. The use of metric results in this way demonstrate another degree of openness, in addition to the testbed's development being influenced by external factors (such as new techniques and contributors) the results it will gather will also influence its development and analysis performed.

5. FUTURE WORK

This initial testbed implementation is merely a first step towards a complete AOSD testbed. Clearly, by using only one benchmark system is one limiting factor. It is also desirable to expose the testbed to new case studies, aggregate new metrics, and embrace benchmarks from varied application domains, such as middleware platforms, pervasive computing, and product lines. It is also the aim of this initial testbed version to generate contributions from the software engineering community to provide new case-studies and benchmarks to develop the testbed in new and different ways.

REFERENCES

[1] S. Soares, P. Borba, E. Laureano. Distribution and Persistence as Aspects. Software: Practice & Experience, 36(6), 2006.

[2] Chidamber, S. and Kemerer, C. A Metrics Suite for OO Design. *IEEE Trans. on Soft. Eng.,20-6*, June 1994, 476-493.

[3] C. Sant'Anna, et al. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. Proc. Brazilian Symp. on Software Engineering, 2003, 19-34.

[4] C. Sant'Anna, et al. On the Quantitative Assessment of Modular Multi-Agent System Architectures. NetObjectDays, 2006.

[5] R. Ramos et al. Evaluation of a methodology for the measurement of the quality of Aspect-Oriented Requirements Document. WER 2005: 161-172.