On the Contributions of an End-to-End AOSD Testbed[1]

[1]Phil Greenwood, [1]Alessandro Garcia, [1]Awais Rashid, [1]Eduardo Figueiredo, [1]Claudio Sant'Anna, [1]Nelio Cacho, [1]Americo Sampaio, [2]Sergio Soares, [3]Paulo Borba, [3]Marcos Dosea, [3]Ricardo Ramos, [4]Uira Kulesza, [5]Thiago Bartolomei, [6]Monica Pinto, [6]Lidia Fuentes, [6]Nadia Gamez, [7]Ana Moreira, [7]Joao Araujo, [8]Thais Batista, [8]Ana Medeiros, [8]Francisco Dantas, [8]Lyrene Fernandes, [9]Jan Wloka, [10]Christina Chavez, [11]Robert France, [12]Isabel Brito

[1]Lancaster University, [2]Pernambuco State University, [3]Universidade Federal de Pernambuco, [4]PUC-Rio, [5]University of Waterloo, [6]Universidad de Málaga, [7]Universidade Nova de Lisboa, [8]State University of Rio Grande do Norte , [9]Fraunhofer FIRST, [10]Universidade Federal da Bahia, [11]Colorado State University, [12]Instituto Politécnico de Beja

Primary Contact: Phil Greenwood (greenwop@comp.lancs.ac.uk)

Abstract

Aspect-Oriented Software Development (AOSD) techniques are gaining increased attention from both academic and industrial organisations. In order to promote a smooth adoption of such techniques it is of paramount importance to perform empirical analysis of AOSD to gather a better understanding of its benefits and limitations. In addition, the effects of aspect-oriented (AO) mechanisms on the entire development process need to be better assessed rather than just analysing each development phase in isolation. As such, this paper outlines our initial effort on the design of a testbed that will provide end-to-end systematic comparison of AOSD techniques with other mainstream modularisation techniques. This will allow the proponents of AO and non-AO techniques to compare their approaches in a consistent manner. The testbed is currently composed of: (i) a benchmark application, (ii) an initial set of metrics suite to assess certain internal and external software attributes, and (iii) a "repository" of artifacts derived from AOSD approaches that are assessed based on the application of (i) and (ii). This paper mainly documents a selection of techniques that will be initially applied to the benchmark. We also discuss the expected initial outcomes such a testbed will feed back to the compared techniques. The applications of these techniques are contributions from different research groups working on AOSD.

1. Introduction

AOSD techniques are increasingly becoming recognized in both academic and industrial circles as useful practices to improve the modularization of artifacts at all software development stages. In addition to the implementation phase, AOSD methods targeted requirements engineering, architecture design and detailed design phases. Regardless of the specific methodology and the targeted development phase, it is vital to perform empirical evaluation to assess the positive and negative effects of AOSD according to different criteria. Furthermore, there is a need to determine the influences of AOSD abstractions used in one particular development stage on subsequent stages. For example, analyzing how candidate aspects indentified during requirements analysis influence the architecture design, what aspects "emerge" or "disappear" and why?

Currently, the majority of previous AOSD assessment studies have considered each phase in isolation, bearing little correlation to preceding and subsequent phases. This limits the conclusions that can be drawn on the effectiveness of AOSD as a whole. The purpose of the testbed is to help answer questions regarding the effectiveness of AOSD throughout the development life-cycle by: (i) providing a set of core applications from different domains in which a variety of software engineering approaches can be applied, (ii) defining metric suites to facilitate end-to-end software lifecycle assessment under different quality perspectives, such as software stability [21], modularity [3, 13, 16, 17, 22], and traceability, and (iii) provide a set of artifacts that have been created from applying a variety of AO and non-AO approaches to the applications provided by (i). Initially, only one application and a limited number of approaches and metrics suites are provided. However, all elements are open for further contribution from the software engineering community. For example, a mapping scheme (see Section 3.2) for AO requirement documents has been developed due to the initiation of this testbed.

A number of initial design decisions had to be made in order to create the testbed, such as selecting: (i) a core case study, (ii) initial metrics to be applied at each development phase, and (iii) both AO and non-AO methodologies to be applied at each phase. This paper documents the decisions made during the design of the testbed and also details the various approaches applied. From providing such artifacts and case studies, the testbed will provide proponents of

[1] This paper is an extension of the two-page paper submitted to the 1st Workshop on Assessment of Aspect-Oriented Technologies (ASAT.07), AOSD.07 Conference, Vancouver, March 2007.

software engineering approaches with the ability to easily compare their approach with others. The testbed will hopefully stimulate the application of various AO approaches and enable the results of such approaches to reach a wider audience. Furthermore, the results collected will be able to feed back into the development of the techniques to improve them further.

The remainder of the paper is structured as follows. Section 2 details and justifies the various design decisions made when implementing the testbed. Section 3 describes the various elements of the tested that includes the approaches applied to the testbed and the artifacts produced. Planned future work and developments for the testbed are detailed in Section 4. This section also includes an overview of how the testbed itself can be evaluated. Finally, the paper is concluded with Section 5 that offers a brief summary of the testbed.

2. Design of the Testbed

The testbed itself consists of three core elements (see Figure 1): (i) Applications that contain a variety of crosscutting concerns; (ii) AO and non-AO approaches that are applied to a common application to generate artifacts; (iii) a suite of metrics associated with a variety of internal and external software attributes; and (iv) a set of metric results that have been gathered from applying the metric suites mentioned in (iii) to the artifacts produced. Each of these elements is extendable to include new instances of applications, approaches, and metrics suites. In order to create these initial core elements a number of factors had to be considered when selecting the application, the target AO and non-AO approaches, and what attributes should be measured. The following segments discuss these factors and decisions made.

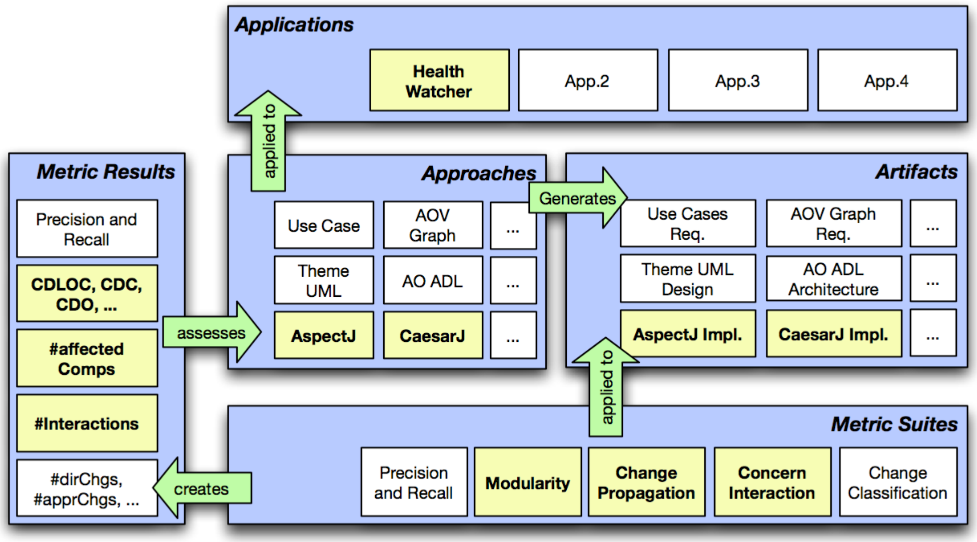

Figure 1. The various elements that compose the testbed.

2.1 Application Selection

Initially, one application was needed that possessed a variety of crosscutting and non-crosscutting concerns in order to allow an assessment of the modularization capabilities of AO and non-AO techniques. Also it was important that the case study was somehow "paradigm neutral", and that any artifacts used to specify the application were designed with modularity, reusability, maintainability, and stability in mind. This was to ensure that any comparisons of AO with non-AO artifacts are fair. Although the application domain was not a factor, it was important that the selected software system was simple to understand, without requiring any specific domain knowledge. Furthermore, the artifacts already created were a factor as it was desirable to select an application where many artifacts were already available (such as a requirements documentation, architectures, and AO and non-AO implementations). The complete set of case study criteria is listed in Table 1.

In all, ten potential candidates were examined, whereby a rating was given for each criterion listed in Table 1. Each criterion was also weighted according to the perceived importance. For example, fundamental characteristics of the applications (such as the types of crosscutting concerns) were deemed more important than more superfluous properties (e.g. the available documentation as new artifacts could be more easily created as needed). The total weighted rating of each application was calculated, and the application with the highest rating was selected. This application was the Health Watcher system [1]; further details are given in Section 3. The inclusion of only one application is severely limiting and does not allow broad conclusions to be made, eventually several applications will be added to the testbed to enable wider conclusions to be drawn. The selection of these additional applications will be based on supplementing certain characteristics of the applications already included in the testbed. For example, applications from different domains will be included to enable broader conclusions to be made. Furthermore, applications should be selected that posses different properties. For example, applications that implement concerns that interact differently or applications that contain concerns that result in architectural ripple-effects.

2.2 Artifact Creation

While applying different approaches and metrics to the initial benchmark application, a range of base artifacts were created. These artifacts are different AO and non-AO representations of the core benchmark application (the Health Watcher system) at a variety of software development stages. The selection of the artifacts and the phases they represent was based on the responses from a questionnaire distributed to a set of AO and non-AO research groups. One of the aims of this questionnaire was to establish the phases where the majority of research was being conducted as so to target the most relevant phases.

Criterion	Purpose	Weight
System generality.	Facilitate benchmark comprehensibility and adoption.	2
Availability of AO and non-AO implementations.	Reduced time in creating the testbed, focus on analysis and result gathering.	3
Availability of complete documentation for development phases other than implementation.	See above.	3
Heterogeneous types of concern interactions.	Provide interesting analysis when artifacts are evolved.	1
Development phase at which aspects emerge (i.e. aspects should appear/disappear at different phases).	Provide better inter-phase analysis.	1
Distinct types of non-crosscutting and crosscutting concerns present in the application (e.g. domain-specific vs. general aspects, widely-scoped vs. localized-scoped aspects)	Prevent bias to one particular set of aspects.	1

Table 1. The criteria for selecting the core application and the associated weighting (1 being the most important).

Furthermore, this questionnaire was used to identify typical quality indicators that are used in each of the phases targeted. For example, Requirements Engineering (RE) approaches commonly make use of precision and recall metrics so it was important that such metrics were included in the testbed. As a result of this questionnaire, the requirements, architecture, design and implementation phases are the ones to be initially targeted in the testbed. This required base artifacts to be created and/or improved at each of these stages including a use-case requirements document, UML-based architecture diagrams, detailed design documentation, and a Java implementation. From these base artifacts additional AO and non-AO approaches can be applied to produce new software artifacts. These can then be compared and assessed with regard the base artifacts and other new artifacts produced from other approaches applied.

2.3 Traceability across Software Engineering Phases

Concern Interaction. As stated in the introduction, one of the key long-term aims of the testbed is to provide support for traceability analysis of modularization techniques across the targeted development phases. This has largely influenced the initial testbed design in the sense that it has required the identification of commonalities across the development phases. For instance, common lifecycle-relevant software attributes and associated metrics. For example, each phase in the development process has some notion of concern interaction. This requires some assessment criteria to be used that can compare and contrast the ways AO and non-AO approaches handle concern interactions across the software lifecycle. Metrics for concern diffusion and concern interaction can be defined for all the software engineering phases.

Modularity Assessment. The distinct lifecycle artifacts share commonalities in terms of modularity attributes. As one of the key aims of AOSD techniques is to improve modularity it is only natural for a testbed to allow this property to be assessed. A series of paradigm-agnostic metrics are also being integrated to the benchmark in order to support modularity evaluation in all development stages. Such metrics have historically been used and successfully validated to assess the modularity of implementation artifacts (e.g. [13, 16, 17, 22]). However, research is being performed to apply similar metrics to requirements [2], architecture [3], and design [4].

Artifact Stability. In addition to performing inter-phase analysis it is also desirable to assess how the AO and non-AO artifacts' modularity is affected in the face of change. This can be achieved by re-applying the modularity metrics after making certain changes to the software artifacts. Applying such metrics to the testbed provides two benefits. First, as each phase has a specific set of modularity metrics, analysis can be performed regarding how the modularity is affected by each change, which offers a further level of "traceability". Secondly, we should recognize that metrics applied to the requirements, architecture and design phases are not mature yet. The testbed is an interesting vehicle to perform case studies in order to both validate and improve such metrics.

Change Impact Analysis. One of the common activities through each of the phases is assessing the impact of change to various properties (i.e. interactions and modularity). To compliment this analysis it is also desirable to directly measure the impact of a change. This involves two branches of assessment, the first assesses the stability of artifacts and the second assesses the amount of effort necessary to perform the change. These two metrics are closely related and one can be inferred from the other. For example, the stability of a use-case document can be measured by the number of use-cases that have not changed or (if fine-grained analysis is needed) the number of steps of a use-case that have not changed. From this, the effort can be inferred and measured in the number of use-cases/steps needed to be altered. Performing comparisons of such data across phases and even artifacts from the same phases can be difficult. The main reason is the presence of differing concepts employed in different phases/artifacts. The analysis performed for interactions and modularity generally uses the concern concepts that are present in all phases so comparisons between phases can be drawn.

2.4 Testbed Contributions

For the testbed to be successful it requires contributions from the software engineering community. Such interest from the community in contributing to this testbed is demonstrated in this paper. Initially, only a limited number of approaches can be applied to the testbed due to time constraints. However, it is hoped that from this first iteration that software engineers can see the benefits from such a testbed and so use it to evaluate any new software engineering approaches developed.

However, for the testbed to expand into a valuable resource it is necessary for developers who use any testbed elements to also contribute back any results or artifacts produced from using the testbed. These submitted results will then become a public resource that can then be used by other software engineers. It can be argued that employing such an approach can lead to inconsistencies and inaccuracies in the artifacts produced and that it would be better for an independent party to apply all approaches. However, this does not take into account the necessary expertise to apply each of the approaches; when each developer is applying his/her own approach we can maximize the chance of the best possible practices being employed.

3. Elements of the Testbed

The purpose of this section is to highlight the various elements present in the testbed. Figure 1 provides an overview of the initial core testbed elements. This figure, involves an overview of the Health Watcher system, initial set of approaches applied to these artifacts, assessment mechanisms applied, and the metric results gathered.

3.1 Health Watcher Overview

The Health Watcher (HW) system [1] is a typical Web-based application that manages health-related complaints. Moreover, it is a real and non-trivial system with existing Java and AspectJ implementations. It encompasses a variety of crosscutting concerns (e.g. concurrency control, persistence and data management, distribution, exception handling) and non-crosscutting concerns (e.g. user interface and business rules). Also, it does not implement complex concepts that are difficult to understand, instead, it uses common technologies such as Servlets, JDBC, RMI. A number of design patterns have been a core part of the original design. In fact, one important property for the testbed is that both non-AO and AO HW designs were equally driven by reusability, maintainability and stability requirements. This ensures that any comparisons between AO and non-AO techniques are fair.

3.2 Requirements Phase

To compare Aspect-Oriented Requirements Engineering (AORE) approaches an initial use-case document was created specifying the behavior of the HW system. A number of conventional and AORE approaches have been systematically applied to the use-case document. Some examples of the AORE techniques applied include:

The viewpoint-based AORE approach encompasses the modularization of aspectual requirements in units called early aspects [5, 6]. The early aspects represent broadly-scoped non-functional properties of the system (e.g. non-functional requirements such as security, performance and compatibility) that impact the core functional requirements of the system expressed in viewpoints. The approach also describes how the early aspects compose with the viewpoints (using composition rules) and how early aspects mutually influence each other that might raise conflicts (e.g., security can negatively influence real-time properties with respect to an ATM viewpoint).

Aspect Oriented Requirement Analysis (AORA) is an aspectual requirements approach to handle separation, modularization, representation and composition of crosscutting concerns [7]. The approach includes a process model, a meta-model to define rigorously the main concepts, and a tool to support the approach. The AORA approach defines three primary tasks, each one divided into several subtasks: identify concerns (i.e. from catalogues such as [8]), specify concerns (to collect information regarding the concern), and compose concerns. This final stage enables the system to be built incrementally and enables conflicts to be identified and resolved using multi-criteria decision methods [9].

Multi-dimensional approach to separation of concerns in requirements engineering (MDSOC) was proposed in [10] as well as trade-off analysis. The work focuses on removing the notion of a fixed functional base with respect to which trade-offs among non-functional concerns are traditionally observed, analyzed and resolved. This approach increases the alignment between the requirements and architecture by treating all concerns, whether functional or non-functional, as peers and thus avoiding architectural decisions being driven solely by functional concerns and increases the ability to perform trade-off analysis.

AOV-graph [27] is an extension to V-graph models in order to avoid the tangling and scattering of crosscutting concerns in requirements models. AOV-graph consists of: i) *goals*, which represent concrete objectives to be achieved for the system; ii) *softgoals*, which represent abstract objectives, frequently associated to non-functional requirements; (iii) and tasks, which represent actions (or functional requirements) to be performed to achieve goals/softgoals. The AOV-graph model also contains the following relationships: (i) correlations, which are used to make explicit the negative and positive influences between goals and softgoals, or softgoals and softgoals; (ii) contributions are used to show how goals/softgoals can be operationalized by tasks, goals/softgoals; (iii) Crosscutting relationships are used to register how softgoals/goals are scattered/tangled through their operationalizations.

The brief description of each of these approaches highlights the significant differences between them. They all introduce different concepts with regard to requirements. As such, assessing and comparing them is difficult. However, certain assessment mechanisms can be used, for example effort/time can be used. For example, applying the MDSOC approach to the base use-case document of the Health Watcher took around 400 minutes. This time can then be compared against the time taken to apply the other approaches.

Furthermore, the testbed is contributing to the development of a common framework for comparing requirements approaches. The framework focuses on comparing these approaches in terms of time effectiveness and accuracy of their produced artifacts. It addresses challenges related to the heterogeneous definitions for AORE model concepts as well as the fact that the AORE approaches perform similar general requirements process activities in different ways. In order to tackle these issues, we provide a mapping of the AORE approaches onto general RE activities and provide a common naming scheme that helps to standardize data collection and comparison of results among the various approaches.

Also, modularization assessment techniques are being developed that can also be applied to requirements documents to evaluate them. The approach described in [11] proposes a set of abstract metrics for this purpose. The abstract metrics, together with some guidelines, can be instantiated to measure requirements documents, obtained using any approach that adopts this paradigm. The guidelines are used to assist the identification of structures and substructures used by an approach to specify and/or modulate a requirement document. These structures are called artifacts. The guidelines help the software engineer describe the general characteristics of the artifacts. The indentified artifacts can then be assessed in terms of: Separation of Concerns (Concern Diffusion over Components and Concern Diffusion over Elements), Size (Vocabulary Size, Number of Elements and Number of Conditional Elements), and Coupling (Coupling between Components) [11].

3.3 Architecture Phase

A measurement framework has been defined for use in the testbed that relies on evaluating the modularization of architectural concerns [3]. It includes metrics for quantifying separation of concerns and their interactions at the architecture design. For instance, it quantifies the diffusion of a concern realization within architecture specification elements, such as components and interfaces. The metrics suite also evaluates how a particular concern realization affects traditional attributes such as coupling, cohesion and interface complexity. Hence it includes metrics for assessing these attributes.

The measurement framework focuses on the evaluation of software architecture representations, such as UML-based or ADL specifications. Also, in order to consider concern as an abstraction in the measurement process, there is a need to explicitly document the concerns in the architecture. Therefore, our approach also includes a notation to support architects with the documentation of the driving architectural concerns [12]. Using this notation, the architect can assign every architecture element (components, interfaces, and operations) to one or more concerns. From applying the proposed measurement framework to existing architectural approaches significant benefits can be gained:

(1) *Use of architectural approaches in more realistic projects.* The testbed provides the appropriate scenario to share and discuss approaches with software engineers, detailed designers and implementers.

(2) *Include architectural approaches as part of a comparison framework between AO software architecture design approaches.* The use of the same case study is fundamental to establish a comparison framework between existing AO software architecture design approaches, especially ADLs.

(3) *Increase the maturity of architectural approaches*. The use of the various architectural approaches as part of the complete AO lifecycle of the case study under consideration will help us to identify and close the gaps between the representation of AO requirements and AO software architectures, as well as between AO software architectures and AO designs.

Similarly to the Requirements phase, a number of non-AO and AO architecture approaches have been applied to the benchmark application. Examples include ACME [25], AspectualACME [23], AO-ADL [26], Aspectual Templates [12], and the AOSD-Europe Visual Notation [28]. Only the AO-ADL and AOSD-Europe notation is described here due to space constraints.

AO-ADL. One of the initial architectural approaches to be applied to the testbed is the AO-ADL which is an XML-based architecture description language specifically suited for modeling AO software architectures. It is targeted at solving the tangled and the scattered behavior problem at the architectural level. The primary contributions will be demonstrated by applying AO-ADL to the testbed applications. The first prominent contribution is the definition of a symmetric composition model, where functional and non-functional concerns are modeled by the same architectural abstraction. Secondly, AO-ADL offers an extension to the traditional semantics of connectors that allow the crosscutting 'aspectual' components to be represented and to specify crosscutting interactions. Finally, AO-ADL defines a catalogue of common crosscutting concerns in the form of connector templates that can be reused in a variety of architectural configurations.

AOSD-Europe Visual Notation. The new version of the AOSD-Europe visual notation [28] defines specific graphical elements to express architecturally-relevant aspectual compositions in component-and-connector models. The new visual elements encompass support for expressing aspectual connectors, crosscutting and base roles, crosscutting relationships, sequencing, and quantification specifications.

When using architectural techniques such as the ones mentioned above there is a need to quantitatively assess the corresponding artefacts to identify, for instance, early modularity anomalies or modelling enhancements. We believe that concern-driven quantitative assessment improves the architecture modularity analysis as it makes more evident the overall influence of the (in)adequate modularization of widely-scoped design concerns.

3.4 Design Phase

The design phase of the testbed is currently under definition. However, a similar approach to the previous phases will be employed, whereby a number of approaches will be applied to specify design documents for the applications used in the testbed. A set of related metrics will also be used to measure such properties as the modularity, design stability and effort to perform maintenance changes. Due to the high similarity between the implementation and design phases it is expected that the implementation metrics (see next section) can be directly reused. However, as the testbed develops and if the need arises, specific design metrics will be introduced.

3.5 Implementation Phase

The implementation phase is the first phase that was developed and carried out in the testbed making the definition of this phase more concrete. As such the configuration used in this phase is highlighted in Figure 1, the elements in the shaded boxes were used in this first study. The implementations of the HealthWatcher System were created using Java, as well as AspectJ and CaesarJ as AO solutions. As the Java and AspectJ implementations already existed, the first phase involved creating an equivalent CaesarJ implementation. By using more than one language yields broader conclusions that are not language specific. Afterwards, the Java, AspectJ, and CaesarJ versions were verified according to the alignment rules, in order to assure a consistent use of code style, functionality and semantics.

The next phase was to implement nine different maintenance scenarios (i.e. the addition/replacement of behavior or the introduction of design patterns to improve the structure) in all three solutions created in the previous phase to assess the stability of each implementation. For each scenario, we needed to ensure again that the Java, AspectJ and CaesarJ implementations were aligned. Each scenario involved the design and implementation of new modules to be included in the system, the use of the language mechanisms to compose such new modules and the existing ones, and if necessary, changes to the modules already existing in the previous HW release.

The quantitative assessment of the design and implementation was based on the application of a metrics suite to both the OO and AO versions of the target system. This suite includes metrics for separation of concerns, coupling, cohesion, and size which have already been used in several experimental studies [13, 16, 17]. The coupling, cohesion, and size metrics are based on traditional metrics, such as lines of code (LOC) [15], and on extensions of widely-used measures for OO design assessment, such as the Chidamber and Kemerer metrics [14]. The metrics suite also encompasses new metrics for measuring separation of concerns [16, 17]. They measure the degree to which a single concern in the system maps to the design components (classes and aspects), operations (methods and advice), and lines of code. The stability of each implementation was also assessed in terms of change propagation, which is measured in terms of the number of components (objects and aspects), operations (methods and advice) and lines of code affected by a particular change. These metrics can also be used to infer the amount of effort necessary to implement a particular change.

4. Testbed Evolution

Initial experiments using the testbed have shown that AOP improves the modularity and leads to more stable designs in several cases [21]. A limitation of these experiments is they only considered the type and number of applied changes, but not the change process itself.

In AO artifacts, pointcuts are particularly considered as fragile (i.e. pointcuts can become incorrect by small changes to artifacts), both at the implementation [18] and requirements [19] phase but also structural adaptation mechanisms can lead to unintended interferences with the program behavior [20]. In future experiments, the analyzability of the change effects will be investigated, taking a change's nature and its dependencies into account. A simple change classification will be used, which considers the effects on other parts of the program and their analyzability. It distinguishes four kinds of effects: direct (symbolic reference, fully resolved names), context-depend (inheritance related effects, structural properties in pointcuts), approximated (dynamic binding, dynamic properties in pointcuts) or indefinite (partially specified name-patterns in pointcuts). The experiments will show how determinable the effects of different changes are.

Although initial set of metric results have been recorded for the requirements and implementation phases, work has still to be carried out at the architecture and design phases. Once this is complete traceability studies will be performed across all development phases. Performing analysis at all development phases will enable a study to be carried out to identify emerging and disappearing aspects. This will allow developers to establish a set of common aspect types that appear at certain development phases. This can aid software development and in particular when mining for aspects.

Work has begun on assessing the usability of the applied RE approaches described in section 3.2. This assessment involved measuring the amount of time spent applying such a technique this can then be used to infer the usability of a particular approach. A similar assessment metric can also be applied to other development phases to assess the usability of the approaches applied, of course using time as an assessment metric might not necessarily be the most appropriate unit of measurement.

5. Summary

This paper has covered the various elements necessary to create an AOSD testbed. This testbed has the goal of facilitating proponents of AO and non-AO approaches to compare and contrast their approaches with others in a more effective fashion. All elements of the testbed are extendable and it is hoped that the software engineering community will contribute various resources (e.g. artifacts generated from new approaches and metric results). One of the key properties of AO approaches the testbed aims to assess is the stability of design. Various typical maintenance changes will be applied to each of the developments phases to assess whether AO techniques contribute to stable designs through the development process. Some of the problems that will be encountered when developing such a testbed have been highlighted including the different concepts various approaches employ. As such it is necessary that the metrics applied take these differences into account and also be paradigm neutral as to not favor AO or non-AO approaches. With this variety of metric results a valuable resource will be created that will easily allow concrete conclusions regarding the contributions of AOSD techniques to be drawn. A further benefit expected to be drawn from the testbed is the provision of data that will allow the tools/approaches applied within the testbed to be improved. Future versions of the developed tools/approaches can be re-applied to the testbed to establish the improvements made.

Acknowledgements. This work is partially supported by the TAO project, funded by Lancaster University Research Committee.

6. References

[1] S. Soares, P. Borba, E. Laureano. Distribution and Persistence as Aspects. Software: Practice & Experience, 36(6), 2006.
[2] R. Ramos and J. F. B Castro. Evaluation of a methodology for the measurement of the quality of Aspect-Oriented Requirements Document. WER 2005: 161-172.
[3] C. Sant'Anna, C. Lobato, Kulesza, U., Chavez, C., Garcia, A, Lucena . On the Quantitative Assessment of Modular Multi-Agent System Architectures. NetObjectDays, 2006.
[4] Chidamber, S. and Kemerer, C. A Metrics Suite for OO Design. IEEE Trans. on Soft. Eng.,20-6, June 1994, 476-493.
[5] A. Rashid A. Moreira, J. Arauho. Modularization and Composition of Aspectual Requirements. AOSD 2003. USA.
[6] A. Rashid, P. Sawyer, A. Moreira, J. Araujo. Early Aspects: a Model for Aspect-Oriented Requirements Engineering. in International Conference on Requirements Engineering (RE). 2002. Essen, Germany: IEEE.
[7] I. Brito, A. Moreira, "Towards an Integrated Approach for Aspectual Requirements", 14th IEEE International Requirements Engineering Conference, Minneapolis, USA, September 2006.
[8] L. Chung, B. A. Nixon, E. Yu and J. Mylopoulos, Non-Functional Requirements in Software Engineering, Kluwer Academic Publishing, 2000. 472pp. ISBN 0-7923-8666-3.
[9] T. L. Saaty, 1980, Analytic Hierarchy Process, McGraw Hill.
[10] A. Moreira, Araújo, J., Rashid, A. "A Model for Multi-Dimensional Separation of Concerns in Requirements Engineering". CAiSE'05, 13-17 June 2005, Porto, Portugal. Lecture Notes in Computer Science, Volume 3520, 2005.
[11] R. Ramos, Araújo, J., Castro, J. F. B., Moreira, A., Alencar, F., Silva, C. "Uma Abordagem de Instanciação de Métricas para Medir Documentos de Requisitos Orientados a Aspectos". In: III Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos WASP´2006. Santa Catarina - Brasil (2006).
[12] A. Garcia, T. Batista, A. Rashid, C. Sant'Anna. Driving and Managing Architectural Decisions with Aspects. SHARK.06 workshop, ICSR.06 Conference, Turin, Italy, June 2006.
[13] N. Cacho, C. Sant'Anna, E. Figueiredo, A. Garcia, T. Batista and C. Lucena. Composing Design Patterns: A Scalability Study of Aspect-Oriented Programming. Proc. AOSD, Germany, 2006.
[14] S. Chidamber, C. Kemerer. A Metrics Suite for Object Oriented Design. IEEE Trans. on Software Eng., 1994, pp. 476-493.
[15] N. Fenton, S. Pfleeger. Software Metrics: A Rigorous and Practical Approach. London: PWS, 1997.
[16] F. Filho, N. Cacho, E. Figueiredo, A. Garcia, C.Rubira. Exceptions and Aspects: The Devil is in the Details. FSE, 2006.
[17] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, A. Staa. Modularizing Design Patterns with Aspects: A Quantitative Study. Trans. AOSD, 1, 2006, pp. 36-74.
[18] M. Stoerzer. and Graf, J. 2005. Using Pointcut Delta Analysis to Support Evolution of Aspect-Oriented Software. Proc. 21st IEEE Intl. Conference on Software Maintenance (ICSM'05), September 25 - 30, 2005. ICSM. IEEE Computer Society.
[19] R. Chitchyan, A. Rashid, P. Rayson, R. Waters, "Semantics-Based Composition for Aspect-Oriented Requirements Engineering", AOSD 2007 (to appear), 2007.
[20] M. Störzer and Jens Krinke, "Interference Analysis for AspectJ", Proceedings of workshop FOAL 2003, held in conjunction with AOSD 2003, 2003.
[21] P. Greenwood et al. On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study. ECOOP, 2007, Germany.
[22] U. Kulesza, C. Sant'Anna, A. Garcia, R. Coelho, A. Staa, C. Lucena. Quantifying the Effects of Aspect-Oriented Programming: A Maintenance Study. Proc. 9th International Conference on Software Maintenance (ICSM'06), Philadelphia, USA, September 2006.
[23] A. Garcia, T. Batista, A. Rashid, C. Sant'Anna. On the Modular Representation of Architectural Aspects. Proc. of the 3rd. European Workshop on Software Architecture, France, 2006.
[24] T. Bartolomei, A. Garcia, C. Sant'Anna, E. Figueiredo. Towards a Unified Coupling Framework for Aspect-Oriented Programming. Proc. 3rd International Workshop on Software Quality Assurance (SOQUA 2006) at FSE.06, Portland, Oregon, USA, November, 2006.
[25] D. Garlan. R. Monroe, D. Wile. ACME: An Architecture Description Interchange Language, CASCON'97, Nov. 1997.
[26] I. Krechetov, B. Tekinerdogan, M. Pinto, L. Fuentes. Initial Version of Aspect-Oriented Architecture Design Approach. 2006, University of Twente: Twente. AOSD-Europe Deliverable D37, AOSD-Europe-UT-D37. p. 1-57.
[27] L. Fernandes, "An Aspect-Oriented Approach to Model Requirements", RE'05 Doctoral Consortium, 2005.
[28] I. Krechetov, B. Tekinerdogan, A. Garcia, C. Chavez, U. Kulesza. Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design. Proceedings of the 8th International Workshop on Aspect-Oriented Modeling, AOSD'06, March 20-24, 2006, Bonn, Germany.