REAL WORLD EVALUATION OF ASPECT-ORIENTED SOFTWARE DEVELOPMENT

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computer Science at Massey University, Palmerston North, New Zealand

CHRISTOPHER MARK ELGAR

2006

Abstract

Software development has improved over the past decade with the rise in the popularity of the Object-Oriented (OO) development approach. However, software projects continue to grow in complexity and continue to have alarmingly low rates of success.

Aspect-Oriented Programming (AOP) is touted to be one solution to this software development problem. It shows promise of reducing programming complexity, making software more flexible and more amenable to change. The central concept introduced by AOP is the aspect. An aspect is used to modularise crosscutting concerns in a similar fashion to the way classes modularise business concerns. A crosscutting concern cannot be modularised in approaches such as OO because the code to realise the concern must be spread throughout the module (e.g. a tracing concern is implemented by adding code to every method in a system). AOP also introduces join points, pointcuts, and advice which are used with aspects to capture crosscutting concerns so they can be localised in a modular unit.

OO took approximately 20 years to become a mainstream development approach. AOP was only invented in 1997. This project considers whether AOP is ready for commercial adoption. This requires analysis of the AOP implementations available, tool support, design processes, testing tools, standards, and support infrastructure. Only when AOP is evaluated across all these criteria can it be established whether it is ready to be used in commercial projects. Moreover, if companies are to invest time and money into adopting AOP, they must be aware of the benefits and risks associated with its adoption. This project attempts to quantify the potential benefits in adopting AOP, as well as identifying areas of risk.

SolNet Solutions Ltd, an Information Technology (IT) company in Wellington, New Zealand, is used in this study as a target environment for integration of aspects into a commercial development process. SolNet is in the business of delivering large scale enterprise Java applications. To assist in this process they have developed a Common Services Architecture (CSA) containing components that can be reused to reduce risk and cost to clients. However, the CSA is complicated and SolNet have identified aspects as a potential solution to decrease the complexity.

Aspects were found to bring substantial improvement to the Service Layer of SolNet applications, including substantial reductions in complexity and size. This reduces the cost and time of development, as well as the risk associated with the projects. Moreover, the CSA was used in a more consistent fashion making the system easier to understand and maintain, and several crosscutting concerns were modularised as part of a reusable aspect library which could eventually form part of their CSA.

It was found that AOP is approaching commercial readiness. However, more work is needed on defining standards for aspect languages and modelling of design elements. The current solutions in this area are commercially viable, but would greatly benefit from a standardised approach. Aspect systems can be difficult to test and the effect of the weaving process on Java serialisation requires further investigation.

Acknowledgements

I wish to acknowledge my supervisors Dr. Jens Dietrich (Massey University) and Shane Griggs (SolNet Solutions Ltd) for their time, support, and ideas during this project. Without them, this project would not have been so successful.

Thank you to Technology New Zealand for their financial support of this project through the Technology in Industry Fellowship (TIF).

To all the staff I have been involved with at SolNet Solutions thank you for your time and feedback as I tried out ideas which will hopefully make their way into your everyday work! In particular, Peter Abbott and Antony Binns for their support when deploying and working with some complex systems, Simon Brierley for embracing AOP and taking it into a real project, and all the Senior Developers and Architects who were interviewed to find out more about how SolNet operates.

Thank you to my parents and relatives for their support during the preparation of this thesis. In particular, your proof reading and comments were greatly appreciated.

To anyone I have missed, apologies, and thank you for your contributions!

Table of contents

List of Figures

List of Tables

Listings

CHAPTER 1
INTRODUCTION

1.1 The Software Development Problem

Software development has long been prone to spectacular project failure rates that would be unacceptable in any other professional discipline. The 1994 Chaos Report from The Standish Group showed that just 16% of projects were successful. Of those unsuccessful projects 31% were never completed and 53% had problems such as cost or time overruns and missing functionality (The Standish Group 1994). An example of a high profile project failure in New Zealand was the Integrated National Crime Information System (INCIS). This ambitious project suffered numerous time delays and cost overruns before it was eventually abandoned with only a small portion of the system in operation (Small 2000).

We believe many project failures can be attributed to the sheer size and complexity of software system developments and the inability of traditional development methodologies to cope with this. Object-Oriented (OO) technology has become the major development methodology helping to reduce complexity with new concepts such as inheritance, abstraction, and polymorphism (Boner, Vasseur & Dahlstedt 2005*a*). The latest Chaos Report in 2003 shows a substantial improvement since 1995 with 34% of projects categorised as successful and only 15% of projects failing. However, 51% of projects still have some problems (The Standish Group 2003). Despite the advances made in recent years, professionals continue to strive to find ways to improve project success rates.

In this thesis, Aspect-Oriented Programming (AOP) is presented as a development approach which has the potential to reduce software complexity and increase software project success.

1.2 Aspect-Oriented Programming

Aspect-Oriented Programming is a relatively new programming paradigm invented at the Xerox Palo Alto Research Center (Xerox-PARC) in the mid nineties by Gregor

Kiczales and his research team. It attempts to reduce program complexity using the notion of separating crosscutting concerns from the core program concerns (Kiczales, Lamping, Mendhekar, Maeda, Lopes, Loingtier & Irwin 1997). This is considered to be one of the most promising approaches to reducing program complexity, and was ranked in the 10 emerging technologies that will change the world by Massachusetts Institute of Technology's (MIT) Technology Review (van der Werff 2001).

AOP adds the concept of an aspect for the purposes of designing and implementing crosscutting concerns. Aspects complement the more familiar concepts of procedures and objects found in the Structured and OO paradigms (Kiczales 2005). AOP is not a replacement for these other paradigms, but rather complements them with a new modularisation technique. Core program concerns can be implemented using traditional modularisation techniques and crosscutting concerns using aspects.

1.2.1 Object-Oriented Programming

Although aspects can be used with other programming paradigms such as Structured Programming, most implementations available are based around current OO languages. The reasons why the OO paradigm is not suitable for all problems faced in modularising code and how aspects complement this technology to solve these problems is discussed. OO was designed to model real-world domain entities and their behaviour as objects. However, there are many elements of a design that must be intermixed with these objects which are incongruent with the object's original intent. AOP addresses this problem by allowing behaviour to be added to objects in a non-intrusive, modularised fashion (Glover 2004).

A good example is a banking system with an 'Account' class containing a 'withdraw' method. Being a banking system there are many things that must happen before and after the 'withdraw' method modifies the account's balance such as security checks, auditing, transaction handling, and persistency. All these extra concerns are not directly part of the main concern of withdrawing funds from an account, but they must be coded with the logic for withdrawing money to ensure the system meets its non-functional requirements. Clearly these extra concerns will require more code than the actual withdrawal of money, and concerns such as transaction handling will be spread across multiple classes making it difficult to maintain and evolve. With AOP it is possible to remove these concerns from the core classes and modularise them as aspects. This will make the system easier to design, code, test, and maintain.

Although the concepts of AOP are not inherently linked to any particular OO language, most of the current mainstream implementations are based around the Java language. This is probably attributable to the strong Java open source community rather than any inherent features of the Java language itself since other languages are having implementations developed such as Python, PHP, C#, Ruby, Perl, and Lisp (Wikipedia 2005*a*). Moreover, AOP implementations based on Java have received strong vendor support from groups such as IBM, BEA Systems, Xerox, and JBoss.

1.3 SolNet Solutions Ltd

1.3.1 Company Profile

SolNet Solutions is an Information Technology (IT) company based in Wellington and Auckland, New Zealand, with approximately 125 staff. Their core business is the delivery of J2EE solutions for large enterprise systems.

1.3.2 Current Development Environment

SolNet have invested substantial time and money into their existing development processes and tools to enable them to produce high quality, reliable systems, as cheaply and timely as possible. SolNet has developed a set of standard reusable components that can be used in typical J2EE projects. These components enable them to significantly reduce the cost and risk involved in conducting J2EE projects. This set of components is referred to as the Common Services Architecture (CSA).

1.3.3 Motivation for AOP Assessment

SolNet's current infrastructure (CSA) is complicated and relies on individual developers being familiar with the components available and how to correctly use them. SolNet are continually looking for ways to reduce the complexity and make their CSA easier for developers to use and more reusable across different types of projects. They would also like to increase flexibility such as having the ability to easily change the components used in a project. For example, changing from EJB Persistency to Hibernate by plugging in a different aspect.

Senior development staff at SolNet have recognised that AOP has potential to simplify their CSA and make it more accessible to different projects. SolNet Solutions entered into this Technology in Industry Fellowship (TIF) project to have an assessment undertaken of AOP technology and how it fitted into their development lifecycle and to assess the potential benefits it could produce in their commercial environment.

1.4 Project Objective and Scope

There is a substantial amount of research being conducted on AOP, and tools are continually being developed. However, the availability of tools does not necessarily mean that AOP is ready to be used commercially (in the real world). To be used commercially there must also be availability of training resources, books, quality assurance tools, integrated development environments, patterns, diagramming techniques, and support infrastructure. Furthermore, the technologies must meet non-functional requirements such as scalability, fault tolerance, and openness.

In this thesis AOP is examined over several of these areas to try and establish its readiness for commercial adoption. Moreover, we try to quantify the commercial benefits of AOP by refactoring a real world project to measure the benefits as a result of using an AOP approach. In doing so we can identify areas where SolNet can benefit from AOP and assess the risks and affect on different areas of their development process.

The objective of this thesis is to show how Aspect-Oriented Software Development can be integrated into a real-world environment at SolNet Solutions with the ultimate goal of assessing the readiness of aspects for use in a commercial environment.

The areas investigated are:

- Approaches to Aspect-Oriented Programming.
- Tool support for Aspect-Oriented development.
- Fitting aspects into the design process.
- Aspect-Oriented standards.
- Testing aspects.
- Metrics for evaluating aspect software.
- Refactoring a real-world project to use AOP.
- Measuring the risks and benefits of using AOP with respect to the refactored project.

Prior to starting this project it was estimated that aspects could reduce the total cost of ownership for SolNet projects by 6%. In this thesis we try to quantify the benefits of AOP and evaluate them against this hypothesis. However, this may not be possible because of the limited historical data available from SolNet to provide a baseline for comparison.

The results obtained are reported in the context of the SolNet Solutions' environment. However, this environment is considered to be representative of many J2EE development companies. It is believed that other companies face similar problems and would have comparable benefits and risks in adopting an Aspect-Oriented approach. Therefore, it is inferred that the results obtained will be applicable to other commercial environments.

Due to the rapidly changing nature of the Aspect-Oriented community, certain limitations were realised when making some assessments. These were made from a practical perspective to enable the work to be completed despite changes happening concurrently with the technologies being evaluated. This was most critical when evaluating the different AOP approaches and tool support. During these two phases the major implementations continually released new versions and features as well as fixing bugs. To continually update and incorporate the new information would have been an endless task. For this reason the current version at the time of conducting the work was evaluated. Some upcoming features are mentioned, but they are not evaluated.

There are many different approaches similar to aspects for achieving separation of concerns such as Composition Filters (Aksit 2001), Hyperslices (Tarr & Ossher 2001), and Subjects (Wikipedia 2005*b*). However, these approaches are outside the scope of this thesis and are only briefly examined.

1.5 Overview of Thesis

This chapter has introduced the objectives and scope of this thesis. In the remaining chapters findings from applying aspects at different places in the software development lifecycle are discussed.

Chapter 2 reviews AOP concepts and terminology for unfamiliar readers.

The different techniques used for AOP are explored in Chapter 3. This compares and contrasts the most popular implementations available. One of the more experimental implementations is examined to see what motivates the development of the smaller frameworks and how their approach differs from the mainstream implementations.

The motivation for Aspect-Oriented tool support is explored in Chapter 4. This includes design, build, development, testing, documentation, and quality assurance tools. The quality of the current tools is assessed and the need for improved tools is identified.

In Chapter 5 the various notations developed to guide the design of Aspect-Oriented software are reviewed, in particular some extensions to the de facto standard Unified Modelling Language (UML) are considered. The use of a notation based on standard UML extensions is proposed. Finally, it is discussed how this notation can be integrated with SolNet Solution's design techniques.

Test driven development has become an important approach for developing quality software. In Chapter 6 the techniques available for testing Aspect-Oriented software are assessed.

To enable us to assess whether our Aspect-Oriented refactoring of a SolNet project has made any improvements, a set of objective, quantitative measurements is required for evaluating AO solutions. In Chapter 7 the use of traditional metrics is proposed. The potential to use new Aspect-Oriented metrics is discussed.

Chapter 8 explores the standards that have been developed for AOP. It then makes recommendations for future standardisation paths to make AOP easier to adopt.

The major goal of this project is to assess the benefits and risks to SolNet Solutions in adopting AOP. Chapter 9 shows the integration of AOP into two real projects which SolNet is undertaking with the New Zealand Qualifications Authority (NZQA). A qualitative and quantitative analysis of the benefits and risks in using AOP for these projects is presented. Finally, further possibilities for utilising AOP at SolNet Solutions are discussed.

Chapter 10 presents a summary of findings from this project and recommendations for future work. The applicability of the findings to environments outside of SolNet Solutions is discussed.