

Aspect Team

Arno Schmidmeier
AspectSoft
Lohweg 9
91217 Hersbruck
Germany

Arno@aspectsoft.com

ABSTRACT

This paper describes a proven approach to the adoption of AOP in commercial projects, which balances the risks and effort on one side versus the benefits on the other. The approach splits the team in two divisions, one focusing on business logic and one on aspects and infrastructure and uses heavily agile planning mechanisms to balance the risk of the implementation of each aspect.

Categories and Subject Descriptors

D.3.3 [Programming Languages]: Language Constructs and Features

General Terms

Management, Economics, Experimentation, Human Factors

Keywords

Agile Processes, AspectJ, Adoption of AOP.

1. INTRODUCTION

AOP is an interesting new programming concept. It has proven its merits in several projects in different sizes. Reductions of more than 50% of the lines of code tend to be the rule, not the exception. [3] However applying AOP is still quite risky due to the usual suspects of new technologies; e.g. increased compositional complexity, immaturity of the implementations, new required tools, additional training costs, missing literature and training opportunities just to name some prominent ones. These effects appear still in the mature AOP language AspectJ [1] or in the mature Spring AOP platform [2]. The paper describes the cookbook recipe aspect team, which provides a convenient way to balance the risks and the drawbacks

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AOSD '07, Workshop BPOAOSD '07, March 12-13, 2007 Vancouver, British Columbia, Canada

Copyright 2007 ACM 1-59593-662-2/07/03... \$5.00

versus the quick benefits of AOP adoption. This pattern has been applied in numerous projects with different sizes (4-50 people) and different software development processes, in which I was involved and responsible to introduce AOP. All projects however had a common property, which was constant integration, at least once a week.

2. Context and symptoms

You run your own software project. Your development process is quite reliable and you do constant integration (at least once a week). The project may be a high risk project, the project may be tight on schedule, staff or money. (And maybe on all of three) You have to deal with typical infrastructure concerns. They may be related to issues of your organisations (e.g. auditing, companies exception policy), related to typical technical issues, (e.g. transaction handling, fault handling, caching, etc.) or even related to constraints imposed by the used framework. (E.g. thread dispatch in GUI frameworks) These concerns result in tangling redundant code. Typical symptoms are:

- long coding instructions, which tell the developer how to code specific code fragments, e.g. how to do exception auditing correctly, what a session bean method has to do, ...)
- excessive use of code templates in the IDE,
- excessive use of cut and paste
- repeating wishes from expert developers to switch to platforms, which offer a coding by convention approach, or to apply a MDS approach.

3. Short Solution Description

In order to apply this recipe, you have to perform these steps:

- Split your team, into a business division and an aspect and infrastructure division.
- Create an aspect candidate backlog (Priorities based on risk, value, effort, etc., similar to a Scrum backlog)
- Let the infrastructure team work on each aspect candidate based on the prioritization,
- Offer the possibility to drop or postpone difficult aspects, by a kind of stop loss policy.
- Whenever an aspect is ready introduce the aspect to the business team and let them decide if they can accept it, if they accept it, set it in the wild.

4. Long Solution Description

First you have to split your team in two parts, one in an infrastructure improvement team and one in business development team. Normally these teams have been named “Aspect and Infrastructure Team” (ait) and “Business Team” (bt). Staff the ait with people interested in technical infrastructure, (50% should be experienced infrastructure aware developers).

Next, you have to perform two steps; both can be normally performed in parallel. You have to set up the AOP infrastructure and you have to create an aspect backlog. The aspect backlog is a aspect implementation scheduling and planning tool. I performed following steps to create the initial base of the aspect backlog:

- List all typical code fragments, which you think, that they could be handled more efficiently by aspects. (Name them aspect candidates) limit the applicability to technical and infrastructure aspects. Write each aspect candidate on a small card.
- Estimate the effort to implement each aspect candidate, estimate the benefit of each candidate, write that information also on the card.
- Estimate the risks of the AOP implementation of each aspect candidate. Involve the infrastructure team in the estimates. (better: make them committed to the estimates). Note this information also on the card
- Prioritise the development of the Aspect candidates based on the ratio between risk, effort to implement and saved effort material. It is fine to reprioritise the aspects in the aspect backlog. Changing the order of the cards means a short reprioritisation.

Setting up the AOP infrastructure, consists of setting up the technical infrastructure and a “knowledge” infrastructure. Setting up the technical infrastructure requires normally a change to the build, deploy and run cycle, so that they use the AOP platform, install and deploy all AOP plugins in the IDE-environment. The knowledge infrastructure consists about a basic understanding of AOP for the whole team. The whole team does need to have a basic understanding about AOP; they need to know what an aspect is, etc. From my experience it is sufficient to transfer this information in a 2-3 hour AOP training session for the whole team. This session should contain:

- Core introduction in AOP (one hour, is enough)
- How to handle the AOP plugins in the IDE (most important, how to detect tangling aspects)

- Questions and answers

After these steps you can start, to work on each aspect candidate from the aspect backlog. I suggest following activities:

- Implement a spike
- If it does work (and prove that the risk and value assumption are fine), implement the aspect, otherwise drop the aspect / put it back in the aspect backlog.
- Create a short aspect presentation (~15 min), (a kind of Aspect Metaphor) which describes the tasks and the prerequisites of the aspect, and how the aspect interacts with the base code. The goal of this short presentation is a kind of aspect awareness by the rest of the team (Note: give the domain team a veto right for unhandable and risky aspects from their point of view)
- branch the code base (We named this branch aspectbranch)
- Integrate the aspect in the code base and maybe clean up the code base,
- Show the system modification to the team
- Merge the aspect branch

Additionally motivate both teams to communicate as easily and as open as possible, and offer common debugging sessions. Aspects candidates do often mushroom during these common debugging sessions.

5. ACKNOWLEDGMENTS

Very much thanks goes to the project teams, which have applied this approach, for their valuable feedback. Additional thanks to my wife and my daughter for being patient, while I am writing papers about my AOP-adoption experiences.

6. REFERENCES

- [1] AspectJ Team, <http://www.eclipse.org/aspectj> (January 2007).
- [2] Interface21, *Spring Framework* <http://www.springframework.org> (January 2007)
- [3] A. Schmidmeier. *Using AspectJ in Component-Based Architectures on the Server Side* Invited Talk, AOSD 2002 Enschede. Slides available from www.aspectsoft.de