An Initial Experiment in Reverse Engineering Aspects

Magiel Bruntink & Arie van Deursen* & Tom Tourwé

Centrum voor Wiskunde en Informatica

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

{*Magiel.Bruntink, Arie.van.Deursen. Tom.Tourwe*} *@cwi.nl*

Abstract

In this paper, we evaluate the benefits of applying aspect-oriented software development techniques in the context of a large-scale industrial embedded software system implementing a number of crosscutting concerns. Additionally, we assess the feasibility of automatically extracting these crosscutting concerns from the source code. In order to achieve this, we present an approach for reverse engineering aspects from an ordinary application automatically. This approach incorporates both a concern verification *and an* aspect construction *phase. Our results show that such automated support is feasible, and can lead to significant improvements in source code quality.*

1. Introduction

In recent years, aspect-oriented software development (AOSD) [1] has become increasingly more popular. AOSD languages offer an additional construct (an *aspect*), which allows software developers to capture crosscutting concerns in cleanly separated modules. The aim is to improve important quality attributes such as maintainability, evolvability and understandability, among others [1, 2].

Up till now, the primary focus of AOSD research was on aspect language technology. Research concerning the migration of large-scale legacy applications to aspect-oriented applications is practically non-existent, even though this is an extremely important and relevant issue. In this paper, our goal is exactly that: we want to assess the feasibility of automatically reverse engineering crosscutting concerns from the source code of a large-scale industrial application. Additionally, this allows us to evaluate whether and how AOSD techniques influence the code quality of such applications.

2. The ASML Case Study

The subject system upon which we perform our experiments is an embedded system developed at ASML, consist-

Concern	Reported Deviations	Unwanted Deviations	% of total
Error handling	18	6	3
Parameter tracing	64	6	1
Parameter checking	75	28	7

Table 1: Results of the concern verification phase

ing of more than 10 million lines of C code. Our experiment, however, is based on a relatively small, but representative, software component (which we will call the *CC* component in this paper).

The crosscutting concerns we focus on for our experiment are the *error handling* concern (responsible for raising, catching and logging errors), the *parameter tracing* concern (responsible for logging the values of input and output parameters of C functions) and the *parameter checking* concern (responsible for implementing pointer checks and raising warnings whenever such a pointer contains a non-expected value). These concerns are implemented by means of simple naming and coding conventions and appear over the entire ASML source code base, comprising roughly 24% of the code of the CC component.

In order to assess the potential benefits of AOSD, we have implemented a *concern verifier* and an *aspect extractor*. The concern verifier automatically analyses the source code and verifies whether it implements the concerns in a correct and consistent way. The term "correct" refers to the fact that the code implementing the concern should adhere to the coding conventions that hold for that concern. The term "consistent" on the other hand, means that the concern should be implemented wherever necessary, e.g. all locations in the source code where the concern should be implemented, implement it. The aspect extractor generates the appropriate aspects corresponding to the way the concerns are implemented in the source code and are verified by the concern verifier.

Parameter	Now	Aspect	% reduced
checking	1441	594	59
tracing	1539	810	47

Table 2: Lines of Code for Parameter Checking and Tracing concern

3. Case Study Experience

Table 1 contains the results obtained from running the concern verifier on the source code of the CC component. The first column of this table represents the concern that is checked, the second contains the number of deviations that were reported, the third contains the number of these deviations that are considered as unwanted, and the last column contains the percentage of the total source code that contains unwanted deviations. This last column may need some clarification:

- for the error handling concern, it states that 3% *of all functions* in the CC component contains one or more unwanted deviations

- for the parameter tracing concern, it states that 1% *of all function parameters* are not traced correctly.

- for the parameter checking concern, it states that as much as 7% *of all function parameters* are not checked correctly.

The results in this table show the advantage of applying AOSD techniques. Although the CC component is quite a small component, it already contains some unwanted deviations. Moreover, we believe these deviations are only due to developer mistakes, because the concerns themselves are quite simple to implement. Such mistakes are primarily due to the fact that these concerns are scattered all over the code, and that despite their simplicity, implementing them requires effort, concentration and discipline.

Table 2 contains a comparison, in terms of lines of code, between the source code as it is now, and as it would be with aspects. Note that our extractor is not yet able to extract the error handling concern, so it is not included in the results. As can be seen, an improvement of 59% and 47% in terms of lines of concern code is achieved when implementing the parameter checking and tracing concerns as an aspect instead of using a simple coding convention. These results indicate that the total amount of lines of code of the CC component can be reduced by 8%.

4. Discussion and Future Work

We have applied the concern verifier and aspect extractor on the CC component only, which is a small, but representative, component in the total ASML code base. In order to confirm the obtained results, we should however ap-

ply these tools to other components as well. Besides ensuring statistically more stable results, this may also lead to an even more refined concern verifier.

Our experiment is only a first, but important, step in a larger effort dealing with the migration of legacy applications to aspect-oriented applications. As such, we have not yet considered important issues such as testing whether the resulting aspect-oriented application preserves the behaviour of the original application, whether automatically transforming millions of lines of C code is feasible, whether adoption of automatic code generators is accepted by developers, and so on. Our long term goal is to investigate all of these issues.

5. Conclusion

In this paper, we have reported on an experiment concerning the verification and extraction of crosscutting concerns in an industrial application. The results of our case study show that automated support for these two activities is both useful and feasible. Furthermore, the benefits from using AOSD technology, compared to simple naming and coding conventions, can be high. We observed that the reduction in amount of lines of code was quite significant, leading to an overall reduction of 8% of lines of code in the considered component.

Acknowledgements

References

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the Europeen Conference on Object-Oriented Programming (ECOOP)*, volume 1241 of *LNCS*, pages 220–242. Springer Verlag, 1997.

[2] R. J. Walker, E. L. Baniassad, and G. C. Murpy. An Initial Assessment of Aspect-Oriented Programming. In *Proceedings of the 21st International Conference on Software Engineering (ICSE)*, 1999.