A SYSTEMATIC REVIEW ON PERFORMANCE EVALUATION OF ASPECT-ORIENTED PROGRAMMING TECHNIQUES USED TO IMPLEMENT CROSSCUTTING CONCERNS

Rodrigo F. G. da Silva[1], Marcelo de A. Maia[1], Michel dos Santos Soares[1]
[1]*Computing Faculty, Federal University of Uberlândia, Brazil*
estudantecomp@gmail.com, marcmaia@facom.ufu.br, mics.soares@gmail.com

Keywords: Aspect-Oriented Programming, Systematic Review, Crosscuting Concerns, Performance

Abstract: Aspect-Oriented Programming (AOP) was proposed with the main objective of addressing an important software quality principle that is modularization. The basic idea of the paradigm is to capture crosscutting concerns as a programming abstraction called aspect. Since the introduction of aspects as a complement to object-oriented programming, many evaluations and empirical studies were provided to the new paradigm, including the application of a variety of software metrics in order to provide evidence of the benefits or problems with the new paradigm. There is no consensus about the impact on performance of the use of AOP techniques to deal with crosscutting concerns. The use of AOP to implement crosscutting concerns and its impact on performance is the motivation for this study. This paper explores further the evaluation of performance by proposing a systematic literature review with the purpose of finding out how performance is affected by the introduction of aspects. The result of this systematic review is that there has been few studies on scientific literature concerning AOP and performance and most of these studies are too specific, and sometimes even inconclusive. This article presents these miscellaneous results and how they were extracted from the literature.

1 INTRODUCTION

Currently, performance is one of the non-functional requirements which has become of utmost relevance for users. Performance is a pervasive quality of software systems (Woodside et al., 2007) and is an important non-functional attribute to be considered for producing quality software (Evangelin Geetha et al., 2011). The performance of software systems is a serious problem in many projects (Smith, 1990), as it may cause delays, cost overruns, failures on deployment, and even abandonment of projects. Even tough, such failures are seldom documented (Woodside et al., 2007).

The growing demand for more complex software which can be executed on several kinds of platforms and architectures, with varied hardware constraints, has been postulating that software can adapt to new requirements, and performance becomes an important feature. This feature, however, can be affected by many elements in a Software Engineering process. One of these elements is related on how crosscutting concerns are implemented in software.

Aspect Oriented Programming (AOP), first presented by Kiczales (Kiczales et al., 1997), came as an attempt to deal with the crosscutting concerns aiming to improve modularization. AOP has gained important interest since its introduction to implement crosscutting concerns, with varying degree of success (Ali et al., 2010) (Przybylek, 2011) (Mortensen et al., 2012). Within AOP, crosscutting concerns are implemented as aspects and are further weaved into code. The way aspects are weaved into code may affect performance as the aspect weaving process introduces new code to the original programs.

There are different approaches to accomplish aspect weaving (Hundt and Glesner, 2009): *Compile-time weaving* which weaves the aspect in a static way into the original code, *Run-time weaving* which weaves the aspect dynamically at runtime, and *Load-time weaving* which delays weaving of crosscutting concerns until the class loader loads the class file and defines it to the Virtual Machine (Dyer and Rajan, 2010).

Because there are different constructions introduced in AOP languages and because there are different ways to implement those constructions, we can expect that a specific running system implemented with AOP may have specific impact in its performance compared for example with traditional procedural languages such as C, which is widely known for producing high performance executables. This concern that also existed when introducing dynamic binding in object-oriented languages, is equally expected to exist in aspect-oriented languages, specially when we consider runtime and load-time weaving, because if the corresponding operation is moved away from the compilation it is expected that it will impact in runtime or load-time.

The impact on performance, caused by AOP techniques, has motivated previous works in scientific literature. Liu (Liu et al., 2011) showed that the aspect-oriented approach does not have significant effect on performance, and that in some cases, aspect-oriented software even outperform the non-aspect one. Additionally, introduction of a large number of join points does not have significant effect on performance. Remko (Bijker, 2005) assessed the performance effects between programs created by a weaver and a hand-coded version. This work came to the conclusion that simple advices give no real performance penalties, but the more sophisticated advices are, the slower they become, and this impact can reach more than 100 percent of penalty. Kirsten (Kirsten, 2005) compared the four leading AOP tools at the time (2004) and, when it comes to performance, he postulated that, in general, code with aspects performs similarly to that of a purely object-oriented solution, where the crosscutting code is scattered throughout the system. Nonetheless, some performance overhead may be noticed either in build-time or in runtime, depending on the used AOP approach. Kirsten also showed the tools' language mechanisms and the trade-offs imposed by the different approaches, as well as the tools' integration with the development environment and build process, including a point-by-point comparison of the tools' IDE features and providing a summary of each one's strengths and weaknesses.

The use of AOP to implement crosscutting concerns and its impact on performance is the motivation for this study. The goal of this systematic review is to understand the extent of the impact of AOP on the performance of software systems, if there is an impact. Our main result reveal that there has been few studies on scientific literature concerning AOP and performance and most of these studies are too specific, and sometimes even inconclusive. There is a lack of studies in which an in-depth focus on performance is taken into account. For instance, there are many variables that can influence the studies that should be taken into account, such as different crosscutting, concerns, different kinds of AOP implementation, different weaving processes, and different tools. In addition, as this research presents, there is no consensus about the impact on performance of the use of AOP techniques to deal with crosscutting concerns. The reason for this is probably because there are too many concerns and several kinds of AOP tools and techniques to be evaluated. This paper shows these miscellaneous results and how they were extracted from scientific literature. We expect to show that this is an open field for further research.

The remainder of this article is organized as follows. In Section 2, we present the research method used for this systematic review. In Section 3, the results are presented and evaluated. In Section 4, we discuss the results and finally in Section 5 we provide the conclusion of this work.

2 RESEARCH METHOD

The main question that motivated this research is: *Does the use of aspect-oriented techniques to implement crosscutting concerns impact software performance ?* A derived research question is "*If the impact exists, how meaningful is it ?*". The answer to both questions could help developers to reason about the feasibility of the use of AOP techniques to handle crosscutting concerns on architectures where performance is itself a concern, as in embedded platforms.

In order to answer both questions, a systematic literature review (Kitchenham, 2004) has been made with the purpose of identifying what type of research has been performed relating AOP and performance. The systematic review started with searching in a number of software engineering conferences and journals. The search was performed considering publications in the past 6 years. Although the seminal works on aspects were published in the late 1990's, we want to evaluate most up-to-date articles, therefore only the last 6 years were considered.

The chosen conferences were: AOSD (International Conference on Aspect-Oriented Software Development) and ICSE (International Conference on Software Engineering). Papers published on specific workshops held together with these conferences were not considered. The chosen journals were: JSS (Journal of Systems and Software), IST (Information and Software Technology), SCP (Science of Computer Programming), TSE IEEE (IEEE Transactions on Software Engineering), TOSEM (ACM Transactions on Software Engineering Methodology). ENTCS (Electronic Notes in Theoretical Computer Science), which can be considered a series, was also included.

The search string used was ("Aspect-oriented pro-

Table 1: Search Results and selected papers

Publication	Retrieved Papers	Relevant Papers	Selected Papers	Data Source
JSS	38	2	1	ScienceDirect
IST	32	10	5	ScienceDirect
SCP	32	2	1	ScienceDirect
TSE	1	1	1	IEEExplorer
TOSEM	21	3	1	ACM Digital Library
ENTCS	26	3	1	ScienceDirect
AOSD	127	9	3	ACM Digital Library
ICSE	61	2	2	ACM Digital Library
Total	338	32	15	

Table 2: Selected papers

	Venue/year	Reference
1	ICSE/07	(Froihofer et al., 2007)
2	SCP/08	(Fabry et al., 2008)
3	IST/09	(Georg et al., 2009)
4	ENTCS/09	(Hundt and Glesner, 2009)
5	IST/09	(Ganesan et al., 2009)
6	ICSE/09	(Zhang, 2009)
7	AOSD/09	(Cannon and Wohlstadter, 2009)
8	JSS/10	(Malek et al., 2010)
9	ACM/10	(Dyer and Rajan, 2010)
10	IST/10	(Ortiz and Prado, 2010)
11	AOSD/10	(Toledo et al., 2010)
12	IST/10	(Janik and Zielinski, 2010)
13	AOSD/10	(Ansaloni et al., 2010)
14	IEEE/12	(Mortensen et al., 2012)
15	IST/12	(de Roo et al., 2012)

gramming" AND "performance"). The search has retrieved 338 papers. From these 338 papers, a sub selection has been made with the purpose of separating those relating AOP with any performance metrics. In a first step, 32 papers were selected and classified as relevant. For the first selection, the title, the keywords and the abstract were read. If the subject was pertinent to AOP and performance, the introduction and the conclusion were read as well. In case of doubt about the relevance of the paper, specific keywords were searched in the paper, such as *aspect*, *crosscutting* and *performance*. There were also relevant papers which used other terms, including *cost*, *payload* and *overhead* when considering assessment of performance of some AOP technique, and in those cases, they were selected too.

In a second step, of the 32 relevant papers, only those ones which assess the performance of implementation of some crosscutting concern were selected to be fully read. As a result, 15 papers were selected in total. Several types of concerns have been classified by the papers as crosscutting concerns, even though some of them were domain specific. However, papers which had crosscutting concerns implemented through some AOP technique but which did not consider any assessment of the used technique(s), or this assessment was incomplete, were discarded. The summary of the filtering process can be seen in Table1 and the final selection of papers is presented in Table 2.

3 EVALUATION OF RESULTS

All 15 selected papers were fully read for the evaluation. The selected papers were evaluated based on two sets of criteria: Application Type and Performance. This section classify the papers for both sets of criteria.

3.1 Application Type Criteria

The first set of criteria concerns about Application Type and encompass the following metrics: number of assessed studies, lines of code (Size, in LOCs), original programming language (Original PL), aspect

Table 3: Summary of studies

Application Type	Article	Assessed studies	LOC / Size	Original PL	Aspect PL	Application Domain
Middleware	(Malek et al., 2010)	2	NA	Java	AspectJ	Generic
	(Zhang, 2009)	3	12.7 KLOC, 113Kb, 190Kb		FlexSync	Industrial
Embedded	(Hundt and Glesner, 2009)	1	NA		ObjectTeams, Java	
	(de Roo et al., 2012)	2	NA	GPL	NA	
System or Application	(Janik and Zielinski, 2010)	1	NA	Java	JBoss AOP	Generic
	(Ganesan et al., 2009)				AspectJ	Office
	(Cannon and Wohlstadter, 2009)	1	46KLOC		Java, AspectJ	Generic
	(Fabry et al., 2008)	1	NA		KALA	Bank
	(Froihofer et al., 2007)				AspectJ, JBoss AOP	Industrial
	(Mortensen et al., 2012)	3	1.6 KLOC, 13.9 KLOC, 51.6 KLOC	C++	AspectC++	
Language	(Toledo et al., 2010)	1	118 KLOC	JavaScript	AspectScript	Generic
	(Dyer and Rajan, 2010)	2	NA	Java	NA	Industrial
Framework	(Ansaloni et al., 2010)	1			Compatible with AspectJ	Generic
Platform	(Georg et al., 2009)			NA	NA	E-commerce
Web Service	(Ortiz and Prado, 2010)			Java	AspectJ	

programming language (Aspect PL) and application domain.

The application type is related to the type of application of the case studies or experiments which have been assessed by the papers. The following types were retrieved from the papers: Middleware, Web Service, Embedded, Platform, System or Application, Language or Extension (Language) and Framework. Cases where their case studies where described as Monitoring Systems were classified as System or Application. Papers which did not mention the application type of their experiments have been classified under the closest definition of these ones already mentioned.

The number of assessed studies indicates only those studies that were implemented by some AOP technique and were assessed by some kind of metric.

Concerning the lines of code metric (LOC), papers showed LOC in different ways: SLOC (Source lines of code), NCLOC (Non-Comment Lines of Code) and only LOC where no citation about source of comments were made. Some papers showed size of the applications instead of LOC. There were papers, however, which did not present any size or LOC of their studies.

Several languages were identified in the papers concerning the original languages and aspect languages. Some papers presented their own languages or extensions in spite of using the most common programming and aspect languages.

The application domain includes: e-commerce (E-C), industrial application (Ind), Office, Bank and Generic. Cases where there is no specific domain, for example a toolkit or a language extension, were classified as Generic. Some papers, mostly in application type, did not mention the application domain and were also classified by proximity.

The summary of studies is presented in Table 3. Cases where no metric was presented or in which it was not possible to identify were classified as not available (NA).

3.2 Performance Criteria

The second set of criteria concerns Performance. Four metrics were extracted from the papers: weaving type, implemented crosscutting concerns, used performance method and performance overhead.

The weaving type indicates the type of weaving performed by the studies in the papers. Two main kinds were considered: Compile-time and Runtime weaving. Some papers presented studies by perform-

Table 4: Performance analysis of studies

Weaving Type	**Article**	**Implemented crosscutting concerns**	**Performance Method**	**Performance overhead**
Run-time	(Malek et al., 2010)	Stylistic	ext, avmo	ext: -negl, avmo: +1.03x to 1.1x
	(Zhang, 2009)	Synchronization	bos	negl
	(Hundt and Glesner, 2009)	Maintainability, Extensibility and Reusability	ext	- 2x for 100 instances
	(Janik and Zielinski, 2010)	Reconfigurability	ext, cpu, avmo	ext: +1.1x to 1.22x, cpu: +NA, avmo: +NA
	(Ganesan et al., 2009)	Monitoring	Qualitative Observation	not notably
	(Cannon and Wohlstadter, 2009)	Security	pat	+ up to 1.16x
	(Toledo et al., 2010)	Expressiveness	met, cpu	met: + up to 16.1x, cpu: negl
	(Fabry et al., 2008)	Transaction Management	NA	+ NA
Compile-time	(Mortensen et al., 2012)	Caching, CheckFwArgs, Excepter, Singleton, Tracing, CadTrace, FwErrs, FetTypeChkr, Timer, UnitCvrt, ViewCache, ErcTracing, QueryConfig, QueryPolicy	ext, avmo	ext: + up to 1.18x, avmo: + up to 1.15x
	(Ansaloni et al., 2010)	Comunication between threads	ext	+ factor up to 31.08x
	(Ortiz and Prado, 2010)	Device adaptation	ext	negl
Compile-time / runtime	(Froihofer et al., 2007)	Constraint validation	ext	+ varies according to approach
Domain Specific	(Dyer and Rajan, 2010)	Cache	met	+ 1.015x
NA	(de Roo et al., 2012)	Safety	NA	+ NA
	(Georg et al., 2009)	Security	ext	+ varies according to approach

ing Load-time weaving process and these cases were classified as Run-time weaving, as Load-time is a specific stage of Runtime.

Several kinds of crosscutting concerns were retrieved from the papers. There were cases where the crosscutting concerns were domain specific. Papers which treated only one concern in the study prevailed, but there were cases where more than one concern was considered, one of them domain specific (Mortensen et al., 2012).

The used performance methods retrieved were the measurements of running or execution time (ext), business operations per second (bos), average memory overhead (avmo), CPU usage (cpu), qualitative observation of the overall execution (obs), parsing time (pat) and average number of method calls per second (met). Some papers presented more than one assessed variable. In these cases, when there was performance reduction, the considered measurements were based on the worst case. The performance overhead was measured in factors (when comparable to the original implementation) or percentage. Some cases related this overhead as negligible (negl).

The results of the performance assessment performed by the papers are presented in Table 4. In the Performance Overhead column, the "+" and the "-'' signs means decrease and increase in performance, respectively. If a sign is followed by negl, it means that the paper reported a degradation or gain in performance, but this result is negligible according to the authors.

Table 5: Application Type versus Performance

Application Type	**Article**	**LOC / Size**	**Original PL**	**Aspect PL**	**Application Domain**	**Perf. Overhead**
Middleware	(Malek et al., 2010)	NA	Java	AspectJ	Generic	ext: -negl, avmo: +1.03x to 1.1x
	(Zhang, 2009)	12.7 KLOC, 113Kb, 190Kb		FlexSync	Industrial	negl
Embedded	(Hundt and Glesner, 2009)	NA		ObjectTeams, Java		- 2x for 100 instances
	(de Roo et al., 2012)	NA	GPL	NA		+ NA
System or Application	(Janik and Zielinski, 2010)	NA	Java	JBoss AOP	Generic	ext: +1.1 to 1.22x, cpu: +NA, avmo: +NA
	(Ganesan et al., 2009)			AspectJ	Office	not notably significant
	(Cannon and Wohlstadter, 2009)	46KLOC		Java, AspectJ	Generic	+ up to 1.16x
	(Fabry et al., 2008)	NA		KALA	Bank	+ NA
	(Froihofer et al., 2007)			AspectJ, JBoss AOP	Industrial	+ varies according to the approach
	(Mortensen et al., 2012)	1.6 KLOC, 13.9 KLOC, 51.6 KLOC	C++	AspectC++		ext: + up to 1.18x, avmo: + up to 1.15x
Language	(Toledo et al., 2010)	118 KLOC	JavaScript	AspectScript	Generic	met: + up to 16.1x, cpu: negl
	(Dyer and Rajan, 2010)	NA	Java	NA	Industrial	+ 1.15x
Framework	(Ansaloni et al., 2010)			Compatible with AspectJ	Generic	+ factor up to 31.08x
Platform	(Georg et al., 2009)		NA	NA	E-commerce	+ varies according to the approach
Web Service	(Ortiz and Prado, 2010)		Java	AspectJ		negl

4 DISCUSSION

Considering the fact that all the papers selected in this systematic review were fully read, and all the 15 selected ones are about the implementation of cross-cutting concerns through AOP techniques and performance, important results can be extracted from this research.

4.1 On the Target Applications

From the first set of criteria, related to Application type, it is possible to conclude that most papers, 10 out of 15, assessed only one study or experiment. However, only four of them showed the LOC or size of their assessed studies. The two studies that have evaluated more systems, evaluated three small-scale systems (at most 50KLOC or 190Kb). The larger evaluated system had 118KLOC. One hypothesis for such lack of large scale studies is that aspect-oriented programming is not extensively adopted such as object-oriented programming or procedural programming. Therefore, the low adoption from the community restricts the availability of large systems for experimentation.

The prevailing application type was System or Application. That is reasonable to expect because in general this kind of applications are more frequent and more accessible. The prevailing application domain was Industrial applications followed by applications with no specific domain, hereby classified as Generic. We can observe that there is reasonable variability in terms of Application Type and Application Domain.

In Table 5, we combined the application type features and the performance result in order to evaluate if there is some influence of the application type in the performance.

We could observe that in Middleware software the overhead was negligible. In the category System or Application, there is a tendency of more impact in the performance.

The LOC seems not to influence because the larger studies systems had negligible impact on execution time and CPU performance. The Application Domain also seems not to influence the performance because of the high variation in the results. The application domain did not present a clear influence in the performance. The *Industrial* domain, which has the larger number of studies, had also presented negligible and positive impact in the performance.

Finally, concerning the implemented crosscutting concerns in the applications, there was no prevailing concern in the studies, and surprisingly, none of the studies implemented common concerns such as Logging or Exception Handling. This can be an indicative that the studied cases were not representative in terms of typical aspect-oriented software.

4.2 On the Used Programming Languages

The prevailing original programming language was Java with 11 out of 15 studies and the prevailing aspect programming language was AspectJ with 6 out of 15 studies. Here we can see that although Java is the preferred target for aspectizing and AspectJ is the preferred solution for aspects (concerning studies on performance evaluation), we could observe studies on less common solutions, such as FlexSync, ObjectTeams and KALA. This is a negative point for those studies because the underlying aspect technology may not be an adequate representative of the common practice. JBoss AOP was present only in two cases, and Spring AOP was not used in any of them.

Considering the impact of the programming language in performance, we can observe that the original programming language that has significant number of studies is Java, but there was no clear indication that Java is an influence factor. In the same way, AspectJ, which is the prevailing aspect language has shown no direct influence on the performance because it presented either negligible or positive impact in performance. Although only two studies were carried out with JBoss AOP, both studies have shown a positive impact in performance.

4.3 On the Type of Weaving

From the second set of criteria, it is possible to conclude that run-time is the most common weaving type process presented by papers in the experiments. One of the reasons for this choice, instead of compile-time weaving, is the fact that runtime weaving allows aspects to be added to the base program dynamically, which is better for a context-aware adaptation of the applications (Hundt and Glesner, 2009). Some papers not only used the runtime weaving process but also extended it to adequate the process to their studies. Also concerning the weaving process, the papers in general postulated that runtime weaving requires more effort at runtime, impacting on performance, but no proofs about this assumption was found in this research.

4.4 On the Experimental Setting

Papers assessed their experiments in different ways, but the prevailing performance metric was measurement of execution time (ext). The performance overhead varied according to a set of variables such as the used approach, implemented concern, the aspect programming language, weaving type process and the used aspect weaver. This motivates the fact that to reason about AOP and performance is not a trivial task, and further research is necessary. There are many variables that impact performance and a controlled experiment is necessary to understand the impact of each variable. The difficulty of this kind of experiment is its multi-factor nature and in order to control the factors, a factorial design is required. The advantage of multi-factor experiments is that all paired interactions can be analyzed. However, the number of runs grows exponentially with the addition of factors.

Moreover, we know that one important and critical factor for performance evaluation is the workload used for measuring the execution (Jain, 1991). In this case, the workload is strongly influenced by the target application, which defines several other sub-factors: the kind of join points, pointcuts and advices, the ratio of occurrence of AOP constructs and the other non-AOP constructs, the requirement of the application for specific type of weaving. Considering the space for combinanation of levels for these factors, it is challenging (if not impossible) to find a real world application (or a set of them) that can have all the possible levels. So, an alternative could be the design of a synthetic application that could be use as a benchmark for performance evaluation of AOP techniques. This benchmark would need to be meaningful to mimic real world scenarios and would need to be comprehensive to guarantee that all important factors and their respective levels would be considered.

4.5 Threats to Validity

The short number of selected papers after the application of the search string and the criteria to include papers for evaluation is a threat to validity of results. A possible solution could be to include additional venues, and to include workshops in future research. However, in order to assess if the recall of this approach was adequate, we made a query in Google Scholar using the same string ("Aspect-oriented programming" AND "performance") . The query returned 10,400 results, but the best ranked links did not showed relevant results for this research. Next, we decided to restrict the string only to the title of the paper and only 12 results were returned. We analyzed each one of the results and only (Liu et al., 2011) was a new result that have not already been selected. Therefore, we conclude that our recall is fairly adequate.

5 CONCLUSION

This work showed through a systematic review and the further analysis of the retrieved papers that there are few experiments concerning AOP and performance in scientific literature. More specifically, too few experiments were reported about the performance of AOP techniques when implementing crosscutting concerns. From the results, it is clear that there is no prevailing implemented concern in the studies. On the contrary, most of the implemented concerns were domain specific.

Most papers postulated that runtime weaving requires more effort at run-time, impacting on performance. However, according to the results of this research, the weaving type process does not appear to be the only factor that impacts performance. Other variables such as the used aspect weaver, the implemented crosscutting concern, the aspect programming language and even the performance method of evaluation could impact the performance results.

This systematic review about AOP and performance address several research fields for future works. We suggest that one of the possible reasons that explain why performance evaluation of AOP did not attract more attention from the community is rooted in the complexity of establishing a comprehensive design of experiment that could produce more solid analysis on the impact of AOP in the performance of running systems. For instance, the development of synthetic benchmarks would help to explain the impact that transformed crosscutting concerns through AOP have in real applications and in what circumstances that impact occurs in the overall performance.

6 ACKNOWLEDGMENTS

The authors would like to thank CAPES (www.capes.gov.br), CNPq grant 475519/2012-4 , FAPEMIG grant APQ-2086/2011 for the financial support.

REFERENCES

Ali, M. S., Ali Babar, M., Chen, L., and Stol, K.-J. (2010). A Systematic Review of Comparative Evidence of Aspect-Oriented Programming. *Information and Software Technology*, 52:871–887.

Ansaloni, D., Binder, W., Villazón, A., and Moret, P. (2010). Parallel Dynamic Analysis on Multicores with Aspect-Oriented Programming. In *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, AOSD '10, pages 1–12, New York, NY, USA. ACM.

Bijker, R. (2005). Performance effects of Aspect Oriented Programming. Technical report, Twente University, Enchede, The Netherlands.

Cannon, B. and Wohlstadter, E. (2009). Enforcing Security for Desktop Clients using Authority Aspects. In *Proceedings of the 8th ACM international conference on Aspect-oriented software development*, AOSD '09, pages 255–266, New York, NY, USA. ACM.

de Roo, A., Sozer, H., and Aksit, M. (2012). Verification and Analysis of Domain-Specific Models of Physical Characteristics in Embedded Control Software. *Information and Software Technology*, 54(12):1432–1453. Special Section on Software Reliability and Security.

Dyer, R. and Rajan, H. (2010). Supporting Dynamic Aspect-Oriented Features. *ACM Transactions on Software Engeneering Methodology*, 20(2):7:1–7:34.

Evangelin Geetha, D., Suresh Kumar, T., and Rajani Kanth, K. (2011). Predicting the Software Performance During Feasibility Study. *IET Software*, 5(2):201–215.

Fabry, J., Tanter, É., and D'Hondt, T. (2008). KALA: Kernel Aspect Language for Advanced Transactions. *Science of Computer Programming*, 71(3):165–180.

Froihofer, L., Glos, G., Osrael, J., and Goeschka, K. M. (2007). Overview and Evaluation of Constraint Validation Approaches in Java. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 313–322, Washington, DC, USA. IEEE Computer Society.

Ganesan, D., Keuler, T., and Nishimura, Y. (2009). Architecture compliance checking at run-time. *Information and Software Technology*, 51(11):1586–1600. Third

IEEE International Workshop on Automation of Software Test (AST 2008) Eighth International Conference on Quality Software (QSIC 2008).

Georg, G., Ray, I., Anastasakis, K., Bordbar, B., Toahchoodee, M., and Houmb, S. H. (2009). An Aspect-Oriented Methodology for Designing Secure Applications. *Information and Software Technology*, 51(5):846–864.

Hundt, C. and Glesner, S. (2009). Optimizing Aspectual Execution Mechanisms for Embedded Applications. *Electronic Notes in Theoretical Computer Science*, 238(2):35–45. Proceedings of the First Workshop on Generative Technologies (WGT) 2008.

Jain, R. (1991). *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley.

Janik, A. and Zielinski, K. (2010). AAOP-Based Dynamically Reconfigurable Monitoring System. *Information and Software Technology*, 52(4):380–396.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., marc Loingtier, J., and Irwin, J. (1997). Aspect-Oriented Programming. pages 220–242. Springer-Verlag.

Kirsten, M. (2005). AOP@Work: AOP tools comparison, Part 1: Language mechanisms. Technical report, IBM Developer Works.

Kitchenham, B. (2004). Procedures for Performing Systematic Reviews. Keele university. technical report tr/se-0401, Department of Computer Science, Keele University, UK.

Liu, W.-L., Lung, C.-H., and Ajila, S. (2011). Impact of Aspect-Oriented Programming on Software Performance: A Case Study of Leader/Followers and Half-Sync/Half-Async Architectures. In *Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference*, COMPSAC '11, pages 662–667, Washington, DC, USA. IEEE Computer Society.

Malek, S., Ramnath Krishnan, H., and Srinivasan, J. (2010). Enhancing Middleware Support for Architecture-Based Development through Compositional Weaving of Styles. *Journal of Systems and Software*, 83(12):2513–2527.

Mortensen, M., Ghosh, S., and Bieman, J. (2012). Aspect-Oriented Refactoring of Legacy Applications: An Evaluation. *IEEE Transactions on Software Engineering*, 38(1):118–140.

Ortiz, G. and Prado, A. G. D. (2010). Improving device-aware web services and their mobile clients through an aspect-oriented, model-driven approach. *Information and Software Technology*, 52(10):1080–1093.

Przybylek, A. (2011). Impact of Aspect-Oriented Programming on Software Modularity. In *Proc. of the 15th European Conference on Software Maintenance and Reengineering*, pages 369–372.

Smith, C. U. (1990). *Performance Engineering of Software Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Toledo, R., Leger, P., and Tanter, E. (2010). AspectScript: Expressive Aspects for the Web. In *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, AOSD '10, pages 13–24, New York, NY, USA. ACM.

Woodside, M., Franks, G., and Petriu, D. (2007). The Future of Software Performance Engineering. In *Future of Software Engineering, 2007. FOSE '07*, pages 171–187.

Zhang, C. (2009). FlexSync: An Aspect-Oriented Approach to Java Synchronization. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 375–385, Washington, DC, USA. IEEE Computer Society.