A Model Curriculum for Aspect-Oriented Software Development

Johan Brichau, University of Lille
Ruzanna Chitchyan, Alessandro Garcia, and Awais Rashid, Lancaster University
Siobhán Clarke, Trinity College Dublin
Ellie D’Hondt, Vrije Universiteit Brussel
Michael Haupt, Darmstadt University of Technology
Wouter Joosen, Katholieke Universiteit Leuven, Belgium
Shmuel Katz, Technion-Israel Institute of Technology
Jacques Noyé, École des Mines de Nantes
Mario Südholt, INRIA

As new software engineering techniques emerge, there’s a cognitive shift in how developers approach a problem’s analysis and how they design and implement its software-based solution. Future software engineers must be appropriately and effectively trained in new techniques’ fundamentals and applications. With techniques becoming more mature, such training moves beyond specialized industrial courses into postgraduate curricula (as advanced topics) and subsequently into undergraduate curricula.

Aspect-oriented software development (see the sidebar) is one such emerging software engineering discipline that’s transitioning from specialized industrial courses to postgraduate-level teaching in universities.\(^1\) AOSD techniques’ industrial maturity is evident in research\(^2\) and in increasing interest in JBoss AOP (aspect-oriented programming, http://labs.jboss.com/portal/jbossaop), AspectJ (www.eclipse.org/aspectj), and Spring AOP (www.springframework.org/docs/reference/aop.html) for developing Enterprise applications. With AOSD’s increasing maturity comes the need to train developers to think beyond a system’s traditional object-based decomposition. Instead, we must adopt a three-dimensional approach—that is, in addition to object structure and behavior, we must also think in terms of aspects that cut across object boundaries. As a result, any AOSD course must train students in applying this three-dimensional approach throughout system analysis, development, and evolution.

In this article, we propose a model postgraduate curriculum for AOSD that does exactly
that. We derived the curriculum from our practical experiences in running postgraduate courses at nine European institutions. These institutions are part of the European Network of Excellence on AOSD (AOSD-Europe). (Deliverables and reports that describe many of our results and structures are available at www.aosd-europe.net). The curriculum describes the fundamentals of any AOSD curriculum. Its goal is to abstract away from national and geographical boundaries to harmonize AOSD education across institutions.

Such a curriculum has several advantages. First, it provides clear guidelines about the fundamental concepts that should be the focus of AOSD postgraduate education, providing an invaluable resource for anyone designing an AOSD course. Second, it provides a common framework for comparing courses and topic coverage across various institutions. Third, it’s a step toward developing a body of knowledge on AOSD to complement existing related knowledge bases, such as the Software Engineering Body of Knowledge (www.swebok.org).

Model postgraduate AOSD curriculum

We propose an AOSD-Europe curriculum with a modular structure inspired by the software engineering life cycle. This doesn’t imply that we envisage a linear waterfall-style development cycle. Instead, the curriculum offers module boundaries so that an institution can instantiate individual modules even if it isn’t considering a full AOSD course. This, in turn, facilitates curriculum instantiations in varying organizational and educational contexts.

The curriculum consists of six modules. It explicitly forgoes a rigid timetable and instead provides a structured body of knowledge tailored to particular instantiation contexts. Thus, it’s possible to choose specific topics of interest and the teaching mode—from classic lecturing to self-study—on the basis of specific local needs.

Because this is the first complete AOSD curriculum, we can’t compare our approach to others. From our experience, a hybrid pedagogical-andragogical approach to teaching AOSD is most apt. In a nutshell, although some initial old-style lecturing is useful, self-learning initiatives are also essential because (industry) students can focus on specific topics that better relate to their day-to-day development problems.

Aspect-Oriented Software Development

The separation of concerns principle\textsuperscript{1,2} suggests that it’s best to treat each concern (that is, an interest which pertains to a system’s development or its stakeholders) separately from other concerns. The improvements the principle brings forward are rooted in the human cognitive characteristic to better understand and reason about one thing at a time, as well as to lead to simplified project management and parallel development.

Contemporary software development paradigms already use this principle. For instance, in object orientation, applications are modeled and implemented by decomposition of both the problem and solution space into objects, where each object embodies a single concern (such as a person or book). Encapsulation, polymorphism, inheritance, and delegation provide additional support. However, some concerns still remain scattered throughout many different objects because they don’t naturally fit within object boundaries. Such concerns (such as security, mobility, distribution, and resource management) crosscut the other concerns. Aspect-oriented software development techniques facilitate systematic identification, modularization, representation, and composition of such crosscutting concerns. This “aspectification” improves system modularity, creating systems that are easier to develop, adapt, maintain, and evolve.\textsuperscript{3} An aspect consists of elements called advice, which collectively specify the aspect behavior. The other crucial element is join points, which are places in other modules where a given aspect can potentially apply its advice. Typical join points are method execution, object instantiation, or attribute setting. Most aspect-oriented techniques facilitate join point selection via declarative query-like mechanisms. We call such a selection predicate a pointcut expression.

So, AOSD techniques offer abstraction, modularity, and composition support to reason about crosscutting concerns throughout the software life cycle—that is, from requirements engineering to architecture and detailed design to implementation, testing, and evolution. Their benefits include improved ability to reason about the problem domain and the corresponding solution; reduction in application code size, development costs, and maintenance time; improved code reuse; and many others. We can see AOSD’s increasing popularity and industrial application in the involvement of major corporations such as Siemens, IBM, Xerox and Boeing and in MIT’s inclusion of AOSD on its 10 most promising technologies list.\textsuperscript{4} For more industrial applications, see www.jboss.org, www.eclipse.org/aspectj, and www.springframework.org.

References


Module 1: Introduction to AOSD

This module provides a broad overview of aspect orientation. It introduces students to the aspect-oriented paradigm’s origins and
foundations, providing a solid basis and a common terminology to be used in subsequent modules. The fundamental concepts include all major elements of the paradigm: separation of concerns, crosscutting concerns, modularization, aspects, join points, pointcuts, advice, and aspectual composition.

Module 1’s only prerequisite is knowledge of software engineering in some existing, well-known paradigm.

Module 2: Aspect-oriented analysis and design

This module covers a broad spectrum of software development activities, from initial requirements definition to architecture derivation and detailed design production. Each of these life-cycle stages can be realized using various aspect-oriented approaches. This module underlines the problems of tangling and scattering caused by crosscutting concerns in nonaspect-oriented analysis and design approaches. It also presents aspect-oriented approaches for aspect identification, modularization, and composition, using several case studies for illustration. An in-depth experience with a particular analysis and design technique and its related tools is a final important goal of the module. Students achieve hands-on experience of aspect-oriented analysis and design through exercises.

The prerequisites are Module 1 and familiarity with some requirements engineering, architecture, and design approaches. Knowledge of object-oriented (OO) analysis and design techniques (including UML) is desirable.

Module 3: Aspect-oriented programming

Several AOP languages exist today, and most are extensions of existing languages. This module focuses on hands-on experience, giving special care to programming practices in AOP. The module covers various aspect languages, highlighting their differences and commonalities to teach students to abstract from concrete languages and understand aspect orientation’s essential mechanisms. It also touches on implementing aspect language execution models to help students better understand the impact of aspects on program execution (for example, in terms of performance).

The prerequisites are Module 1 and experience in or knowledge about software implementation by means of contemporary languages, preferably in the OO paradigm.

Module 4: Formal foundations of AOSD

This module extends or reevaluates formal notions, such as semantics, specification, and verification, in an aspect-oriented context. It surveys several semantic approaches, concentrating on one or two approaches. It also covers specification of aspects, so that analysis of their desired properties becomes possible. The module presents formal methods for verifying and refining aspect systems, extending classical model checking to aspects, and relating static analysis to classes of temporal properties. The module uses these formal approaches to define and compare declaring and weaving aspects, specify the properties an aspect adds and determine whether these are true when the aspect is woven to a system, and show that composing an aspect doesn’t disturb the system’s desirable properties.

The module also surveys existing work on defining and analyzing interactions among multiple aspects.

The prerequisites include Module 1 and notions of computer science’s formal foundations, especially logic and automata theory. Some instantiations might also require Module 3.

Module 5: Aspect-oriented applications

Module 5 illustrates the practical use of various aspect-oriented technologies, such as programming languages, aspect-oriented analysis and design, and, more generally, any software engineering methodology that embraces aspects. It presents case studies of applications that benefit from AOSD, covering system-level elements (such as middleware) and end-to-end user applications (such as e-banking or e-government applications). The module’s main subject isn’t technologies that create system-level elements and end-to-end user applications; Modules 2 and 3 will have covered these.

The prerequisites are Module 1 and, depending on the instantiation, Module 2 and/or Module 3.

Module 6: AOSD and other paradigms

Aspects are always used in a context. Therefore, to develop applications using aspect-oriented techniques, it’s important to re-
late AOSD to the development paradigms, methodologies, and programming languages that you used to implement the underlying base application. AOSD’s context is almost always class-based object-orientation. However, as AOSD spreads to other contexts, this relationship will diversify and become more important. Also, other advanced development paradigms have been developed that can be related to AOSD, either because they’re complementary or because they target the same problems as AOSD (albeit differently). This module provides insights into the relationship between AOSD and these other advanced development paradigms (for example, development methodologies other than the OO paradigm), different general-purpose programming languages for the base code, and component-oriented software engineering.

The prerequisites are Module 1 and at least one other module.

Curriculum instantiations

A complete instantiation of the AOSD curriculum must incorporate Modules 1 through 6 in some way. We provide several examples below, ranging from a summer school context to university curricula at the master’s degree level. For each, we describe the course context and discuss specific key design challenges. Table 1 shows the course units taught for each instantiation, how they map onto the various modules, and the percentage of classroom hours spent on each module in the program, where available.

Table 1
Curriculum instantiations and their relations to the model curriculum; percentages represent classroom hours spent on each module.

<table>
<thead>
<tr>
<th>Instantiation</th>
<th>Module 1</th>
<th>Module 2</th>
<th>Module 3</th>
<th>Module 4</th>
<th>Module 5</th>
<th>Module 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOSD-Europe</td>
<td>No formal course—students did preparatory reading on their own (0%)</td>
<td>Introduction to AOSD Concepts (10%)</td>
<td>Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design (30%)</td>
<td>Advanced Software-Engineering Techniques; and Advanced Object-Oriented Software Architecture</td>
<td>Advanced Software-Engineering Techniques; and Advanced Object-Oriented Software Architecture; and Object-Oriented Programming Languages and their Implementation</td>
<td>Object-Oriented Programming Languages and their Implementation; Distributed Object-Oriented Systems; Advanced Object-Oriented Software Architecture; and Advanced Software-Engineering Techniques</td>
</tr>
<tr>
<td>Summer School</td>
<td>AOSD</td>
<td>No course—students did preparatory reading on their own (0%)</td>
<td>No course (0%)</td>
<td>No course (0%)</td>
<td>No course (0%)</td>
<td>No formal course—the summer school treats AOSD in the context of OOP (2%)</td>
</tr>
<tr>
<td>Lancaster University</td>
<td>Introduction to AOSD Concepts (10%)</td>
<td>Introducing Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design (30%)</td>
<td>Advanced Software-Engineering Techniques; and Advanced Object-Oriented Software Architecture</td>
<td>Advanced Software-Programming with AspectJ; AOP Models Beyond AspectJ; and Design Patterns Implementations and Assessment of AOP (30%)</td>
<td>Relating Aspects to Proof Obligations (1%)</td>
<td>AOP Application in Persistent, Distributed Environments (20%)</td>
</tr>
<tr>
<td>AOSD</td>
<td>No course—students did preparatory reading on their own (0%)</td>
<td>Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design (30%)</td>
<td>Advanced Software-Programming with AspectJ; AOP Models Beyond AspectJ; and Design Patterns Implementations and Assessment of AOP (30%)</td>
<td>No course (0%)</td>
<td>No course (0%)</td>
<td>No course (0%)</td>
</tr>
<tr>
<td>EMOOSE (the European Master in Object-, Component-, and Aspect-Oriented Software Engineering)*</td>
<td>AOSD</td>
<td>Advanced Software Engineering Techniques; and Advanced Object-Oriented Software Architecture</td>
<td>Advanced Software-Programming with AspectJ; AOP Models Beyond AspectJ; and Design Patterns Implementations and Assessment of AOP (30%)</td>
<td>Advanced Software-Programming with AspectJ; AOP Models Beyond AspectJ; and Design Patterns Implementations and Assessment of AOP (30%)</td>
<td>Relating Aspects to Proof Obligations (1%)</td>
<td>AOP Application in Persistent, Distributed Environments (20%)</td>
</tr>
<tr>
<td>Trinity College Dublin</td>
<td>No course—students did preparatory reading on their own (0%)</td>
<td>No course (0%)</td>
<td>No course (0%)</td>
<td>No course (0%)</td>
<td>No course (0%)</td>
<td>No course (0%)</td>
</tr>
</tbody>
</table>

We don’t mention percentages for each module in Table 1 because they’re hard to assess; the proportion of AOSD-related courses to the total of courses in Emoose ranges from 33 percent to 60 percent, depending on the individual projects’ topics.

November/December 2006 IEEE SOFTWARE 5
The curriculum’s implementation in the examples depends largely on two factors. First, the program context impacts timing and resource issues. For example, a postgraduate program can spread its curriculum over an academic year, whereas a summer school is much more concentrated. Second, instantiations must consider their audience—that is, whether they’re dealing with students, industry-trained attendees, or both. The time the program attributes to each module as well as the application type can differ.

There’s no one right way to implement the curriculum; each instantiation has specific needs. The examples we present here show how you can use the curriculum in varying contexts, providing practical examples for lecturers or departments interested in including AOSD material in their courses.

**The AOSD-Europe Summer School**

The summer school on AOSD (www.aosd-europe.net/summerschool) is a complete, concrete instantiation of the curriculum. Its goal is to provide intensive week-long training on various AOSD facets through lectures by experts from each AOSD subdomain in the curriculum. Typically, each course unit consists of foundational material demonstrated with examples and a group exercise and advanced topics from the lecturer’s research field. Courses include hands-on group sessions where appropriate.

We held the first summer school in July 2006; we’re now processing attendee feedback from a questionnaire. Our main target audience was PhD students.

The program encourages student interaction via a workshop, implemented in the program through a series of research sessions. In these, we invite all summer school attendees to present on their ongoing work. Summer school lecturers provide feedback. This demonstrates perfectly how the curriculum is nonexclusive to additional activities that can reinforce the concepts taught. Also, there’s a focus on AOSD’s key application contexts, particularly on middleware and system software that supports aspect-orientation and on frameworks that model a key application development concern (such as security). This choice is motivated by the fact that both subdomains form a broad area of PhD research topics.

**Intensive course on aspects at Lancaster University**

A second concrete instantiation of the curriculum is a stand-alone week-long intensive course for MSc students at Lancaster University. The course has been running for two years, with an adapted version the second year (this version is now largely stable). The course lasts one week to support the needs of part-time students from industry, which make up about 50 percent of the attendees. Although we utilize traditional lectures, much time is devoted to exercises in various forms. Concretely, we have individual exercises for Modules 1 and 3 (both utilize AspectJ); group exercises for Modules 2, 3, 4, and 6 (some of these utilize AspectJ while others are based on aspect-oriented analysis and design techniques from relevant literature); and a roundtable session on AOP’s pros and cons for Module 5. When the week ends, we expect students to spend two weeks completing an individual project that covers topics from all modules. Although we have no statistical data, student feedback has been unanimously positive.

This course’s main difficulties are in conveying AOSD basics in a way that helps students apply them to realistic industry situations in just one week. First, we must teach AOSD’s fundamental concepts in a short time span and root them in practical examples and hands-on exercises in a concrete programming language framework. We chose AspectJ because it’s mature, has good tool support, and uses most of the conceptual terminology, and our later lectures relate to it. So, we use the practical programming supported approach from early on to both ground the fundamental concepts in aspect-orientation and AspectJ.

Second, a week-long course requires a quick transition from traditional separation of concerns to aspect-orientation—that is, there’s no time for programming concepts to sink in. To address this challenge, we use hands-on tutorials as well as interactive aspect refactoring of Java programs. As such, we can gauge whether students have grasped the fundamentals and can reinforce knowledge where needed.

A third issue is the lack of reading time during the week. To address this, we announce the reading material well in advance of the course, and students partly discuss it in group reading slots and subsequent presentation slots.

Fourth, with an intensive one-week course,
Introducing AOSD within the concept of a full software engineering module has worked very well, giving students time to study, research, and appreciate the overall context.

EMOOSE AOSD minor

EMOOSE (the European Master in Object-, Component-, and Aspect-Oriented Software Engineering) is a yearlong MSc program that began in 1998. It initially focused on OO software engineering, emphasizing up-to-date and future technologies that facilitate software modularization, reuse, and evolution. Although AOSD was present in the program from the start, it became more important gradually.

A new program that includes an AOSD minor began in October 2006. It closely matches EMOOSE’s earlier architecture, with first-semester courses in Nantes followed by a semester-long internship at a partner institution. We divided the course into eight units; those not mentioned in table 1, which don’t relate to Modules 1 through 6, are Fundamentals of OO Technology, Capita Selecta from Computer Science, and a specialization training. Exams and projects assess course knowledge. Apart from Module 1, most modules do not appear explicitly. The minor’s crosscutting nature ensures that students don’t learn AOSD as a stand-alone concept.

Students have evaluated EMOOSE courses. Their comments on the AOSD-minor have shown that they’re satisfied with the new curriculum and they’re more interested in the novel course units.

Integrating a comprehensive AOSD formation in an existing full-fledged yearlong MSc requires a thorough exposition of relationships between AOSD and related domains. Indeed, these form a substantial part of the overall program, which is unlike previous instantiations. We designed EMOOSE course units to make such relationships explicit. Furthermore, in the integrated program, students should adhere to the typical EMOOSE style of strong project-oriented team-based work, mirroring the development of large-scale applications in industry. EMOOSE supports this development with dedicated course units, such as the specialization training, and, more generally, with a course structure that consistently complements lectures with hands-on exercises involving several students.

Finally, the aim is to link theory and practice. That is, even course units that don’t focus on theoretical issues present software construction and validation techniques together with at least minimal information on their foundations.

Trinity College Dublin

Trinity College Dublin’s concrete instantiation of the AOSD curriculum, which has been ongoing for several years, was designed to be incorporated into an MSc module on Software Engineering for Concurrent and Distributed Systems (NDS104). NDS104 is one of six modules in the MSc in Computer Science (Networks and Distributed Systems). We expect students to complete a research dissertation after the teaching terms. NDS104’s philosophy is based on combining research-led teaching with learning through practical application. In the second term, we dedicate four weeks to AOSD topics, with a series of lectures and research readings. We expect students to further research and apply AOSD practices in a large group project.

While student evaluations are at full module level, most feedback on AOSD-related topics seems positive. Moreover, we have had one or two AOP PhD students every year but one.

Introducing AOSD within the concept of a full software engineering module has worked very well, giving students time to study, research, and appreciate the overall software engineering context. The main challenges relate to the expectation that students apply the concepts to a large project.

First, we had to find the right time to introduce AOSD. Although many of our MSc students have significant industrial experience and realize AOSD benefits quickly, many other students don’t have such experience. However, group project work starts early in the year as students are expected to start at the development stage of requirements gathering and analysis. More advanced students start to think about and design the solution early and have commented that they would like to learn AOSD concepts early. We’re now experimenting with this balance.

Second, we aim to have students use the tools in a large project. To date, we have primarily used AspectJ with Eclipse in the proj-
ects. However, as a project’s code base gets larger, students have found the additional build time to be cumbersome and a disincentive to adding new aspects. This academic year, we expect to have at least one project applying MDA techniques and tools, and we’re also interested in how AOP tools perform in that context. We expect, however, that this problem won’t be an issue in the close future because the tools are constantly being improved.

**AOSD curriculum and SWEBOK**

Because AOSD techniques complement existing, established software engineering practices, it’s important to understand the curriculum’s contributions to SWEBOK. We can’t relate our curriculum to bodies of knowledge at the undergraduate level, such as SEEK (Software Engineering Education Knowledge), because instantiations vary from undergraduate courses to advanced industrial courses. It’s reasonable to expect that people with four or more years of practice would attend a course on AOSD—this was the case for 50 percent of the Lancaster institution’s audience. So SWEBOK, a body of knowledge that’s both a learning goal for fresh graduates and a yardstick for established practitioners, is the most suitable means of relaying the contributions of AOSD concepts to general software engineering training and education. Because the curriculum incorporates techniques ranging from aspect-oriented requirements analysis to software systems’ implementation and evolution, it inevitably contributes to all of SWEBOK’s knowledge areas. Table 2 summarizes these contributions.

With experience and lessons learned from these concrete instantiations, additional AOSD-Europe sites have begun instantiating the curriculum modules in their programs by inserting some basic material into industrial training programs and undergraduate courses and by designing courses that crosscut the six modules. This is the case at The Technion-Israel Institute of Technology and also at sites in Belgium that are developing a lightweight school for industry adopters. Furthermore, AOSD-Europe’s long-term existence provides a larger framework for these activities. Within the network, structures are now being conceived that give AOSD-course graduates interesting future possibilities, such as enrolling for an AOSD-Europe-accredited PhD program or continuing to work with industrial partners.

**References**

<table>
<thead>
<tr>
<th>Software Knowledge areas</th>
<th>Module 1</th>
<th>Module 2</th>
<th>Module 3</th>
<th>Module 4</th>
<th>Module 5</th>
<th>Module 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software requirements</td>
<td>Notion of crosscutting requirements</td>
<td>Elicitation, specification, analysis, and validation of aspectual requirements</td>
<td>Implementation of crosscutting concerns using AOP techniques</td>
<td>Deriving proof obligations from aspectual requirements</td>
<td>Concrete applications of aspect-oriented requirements engineering</td>
<td>Relationship with viewpoint-, use-case, and goal-oriented approaches</td>
</tr>
<tr>
<td>Software design</td>
<td>Notion of crosscutting concerns in architecture and design</td>
<td>Early insights into architectural choices based on aspect-oriented requirements analysis; modeling and analysis of aspects in architecture and design</td>
<td>Precisely relating requirements-level aspect trade-offs to architecture and design</td>
<td>Concrete applications of aspect-oriented architecture and detailed design</td>
<td>Relationship with existing ADLs and modeling notations such as UML</td>
<td></td>
</tr>
<tr>
<td>Software construction</td>
<td>Notion of crosscutting concerns in programs</td>
<td>Mapping aspect-oriented designs to implementation</td>
<td>Precise semantics for aspects and aspect composition</td>
<td>Concrete applications of AOP techniques; AOSD in middleware contexts</td>
<td>Relationship with existing paradigms such as OO programming, functional programing, and procedural programming</td>
<td></td>
</tr>
<tr>
<td>Software testing</td>
<td>Deriving test cases and proof obligations from aspectual requirements</td>
<td>Unit testing support for aspects</td>
<td>Modular aspect verification</td>
<td>Aspect testing in concrete application scenarios</td>
<td>Differences between aspect testing and regular testing mechanisms</td>
<td></td>
</tr>
<tr>
<td>Software maintenance</td>
<td>AOSD's evolution benefits</td>
<td>Managing the impact of change to aspects in requirements, architecture, and design</td>
<td>Managing the impact of change to aspects in code; aspect-oriented refactoring</td>
<td>Verifying core properties during aspect maintenance and evolution</td>
<td>Evolvability of aspect-oriented applications</td>
<td>Impact of AOSD on a system's evolution</td>
</tr>
<tr>
<td>Software configuration and management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Relationship of AOSD with software variability management and component configuration</td>
<td></td>
</tr>
<tr>
<td>Software engineering management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Integration of AOSD into existing management practices</td>
<td></td>
</tr>
<tr>
<td>Software engineering process</td>
<td>Integration of aspect-oriented analysis and design into existing software processes</td>
<td>Integration of AOP into existing software processes</td>
<td>Employing AOSD in real-world software engineering processes</td>
<td>Integration of AOSD into existing software processes, such as agile development</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software engineering tools and methods</td>
<td>Tools and methods for aspect-oriented analysis and design</td>
<td>Tools and methods for software verification and code analysis</td>
<td>Applications of concrete aspect-oriented tools and methods</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software quality</td>
<td>Qualitative trade-offs identified from aspect-oriented analysis; improved evolvability, reusability, adaptability, and composability of requirements, architectures, and designs</td>
<td>Improved evolvability, reusability, adaptability, and composability of programs</td>
<td>Consistency management during aspect composition</td>
<td>Empirical evidence of improved software quality with AOSD; metrics for assessing aspect-oriented systems</td>
<td>Comparative improvement in quality attributes offered by AOSD</td>
<td></td>
</tr>
</tbody>
</table>
About the Authors

Johan Brichau is a postdoctoral researcher at the University of Lille as part of the Inria-Jacques project at the Laboratoire d'Informatique Fondamentale de Lille. He's also a postdoctoral researcher at the Vrije Universiteit Brussel’s Programming Technology Lab. His research interests include aspect-oriented programming and early architecture design, multidimensional separation of concerns, and development of composition mechanisms in the requirements and early architecture context. He received his postgraduate diploma in computing from the Open University. Contact him at the Computing Dept., Infolab21, South Dr., Lancaster Univ., Lancaster LA1 4WA, UK; johan.brichau@lifl.fr.

Ruzanna Chitchyan is a researcher in Lancaster University’s Computing Department. She is also a PhD student in computer science at Lancaster University. Her research interests include aspect-oriented requirements engineering and early architecture design, multidimensional separation of concerns, and development of composition mechanisms in the requirements and early architecture context. She received her postgraduate diploma in computer science from Dublin City University. She’s a member of the IEEE. Contact her at the Dept. of Computer Science, Trinity College, Dublin 2, Ireland; siobhan.clarke@cs.tcd.ie.

Ellie D’Hondt is a postdoctoral researcher in the Vrije Universiteit Brussel Brussels Department of Mathematics, funded by the Flemish Fund for Scientific Research. Her research interests include distributed quantum computation and its formal foundations, particularly from the viewpoint of entanglement as a computational primitive. She received her PhD in computer science from Dublin City University. She’s active in the European Network of Excellence on Aspect-Oriented Software Development. Contact her at the Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium; eldhondt@vub.ac.be.

Siobhán Clarke is a senior lecturer in Trinity College Dublin’s Computer Science Department. Her research interests are in service and AOP models for mobile, context-aware systems. She received her PhD in computer science from Dublin City University. She’s a member of the IEEE. Contact her at the Dept. of Computer Science, Trinity College, Dublin 2, Ireland; siobhan.clarke@cs.tcd.ie.

Alessandro Garcia is a computer science lecturer in Lancaster University’s Computing Department. His research interests include empirical evaluation of AOSD, software architecture, exception handling, and agent-oriented software engineering. He received his PhD in computer science from the Pontifical Catholic University of Rio de Janeiro. Contact him at the Computing Dept., South Drive—Infolab 21, Office C-54, Lancaster Univ., LA1 4WA, United Kingdom; garciaa@comp.lancs.ac.uk.

Michael Haupt is a research assistant at the Hasso Plattner Institute for Software Systems Engineering in Potsdam, Germany. His research interests focus on AOSD, virtual machines, and dynamic programming languages, including developing virtual machine support for AOP languages. He received his doctoral degree in computer science from the Darmstadt University of Technology. He’s a member of ACM. Contact him at Software Architecture Group, Hasso Plattner Inst. for Software Systems Engineering, Prof.-Dr.-Heimert-Str. 2, D-14482 Potsdam, Germany; michael.haupt@hpi.uni-potsdam.de.

Wouter Joosen is a professor in the Katholieke Universiteit Leuven, Belgium’s Computer Science Department. He’s also a member of the DistriNet Research Group, which aims at the development of open, distributed object support platforms for advanced applications with a focus on industrial applications. His research interests include developing aspect-oriented middleware. He received his PhD in distributed and parallel systems from the Katholieke Universiteit Leuven, Belgium. Contact him at the DistriNet Research Group, Office A02.31, Celestijnenlaan 200A, B-3001 Heverlee, Belgium; wouter.joosen@cs.kuleuven.be.

Shmuel Katz is a professor in the Technion-Israel Institute of Technology’s Computer Science Department. He’s also the head of the Formal Methods Lab of the European Network of Excellence on AOSD, where he coordinates work on formal methods and semantics for aspects. His research interests include AOP and software development, program verification, partial order reductions in verification, and translations among verification and modeling tools. He received his PhD in program analysis from the Weizmann Institute of Science. Contact him at the Computer Science Dept., Technion, Haifa, 32000, Israel; katz@cs.technion.ac.il.

Jacques Noyé is a lecturer at the École des Mines de Nantes, where he also heads the Master of Science EnWSit (European Master of Science in Object-Oriented and Software Engineering Technologies) and is a member of the Objects, Aspects, and Components group. His research interests include architecture programming languages (particularly component and aspect languages) and the adaptation and specialization of applications based on such languages, using techniques such as reflection and partial evaluation. He received his PhD on variations and extensions of the Prolog Warren Machine from the University of Rennes I. He’s a member of the ACM. Contact him at École des Mines de Nantes, Département Informatique, 4, rue Alfred Kastler, BP 20722 F - 44307 Nantes Cedex 3, France; Jacques.Noye@emn.fr.

Awais Rashid is a faculty member in Lancaster University’s Computing Department, where he leads research in aspect-oriented software engineering. His research interests include early aspects, novel join point models, and aspects in distributed, persistent environments. He received his PhD in computer science from Lancaster University. He also coordinates the European Network of Excellence on AOSD. He’s the author of Aspect-Oriented Database Systems (Springer, 2004) and a founding coeditor in chief of Transactions on Aspect-Oriented Software Development. Contact him at Computing Dept., Infolab21, South Drive, Lancaster Univ., Lancaster LA1 4WA, UK; awais@comp.lancs.ac.uk.

Mario Sudholt is a researcher at Inria. He’s also a lecturer at the École des Mines of Nantes and a member of the university’s Objects, Aspects, and Components group. His research interests include event-based AOP, aspects with explicit distribution, and software components with explicit protocols. He received his PhD in parallel programs from the Technische Universität Berlin. Contact him at École des Mines de Nantes, Département Informatique, 4, rue Alfred Kastler, BP 20722, F - 44307 Nantes Cedex 3, France; sudholt@emn.fr.


For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib