

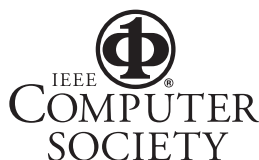


## **A Model Curriculum for Aspect-Oriented Software Development**

*Johan Brichau, Ruzanna Chitchyan, Siobhán Clarke, Ellie D'Hondt, Alessandro Garcia, Michael Haupt, Wouter Joosen, Shmuel Katz, Jacques Noyé, Awais Rashid, and Mario Südholt*

Vol. 23, No. 6  
November/December 2006

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.



© 2005 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

For more information, please see [www.ieee.org/portal/pages/about/documentation/copyright/polilink.html](http://www.ieee.org/portal/pages/about/documentation/copyright/polilink.html).

# 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*

**Shmuel Katz**, *Technion-Israel Institute of Technology*

**Jacques Noyé**, *École des Mines de Nantes*

**Mario Südholt**, *INRIA*

A model curriculum for aspect-oriented software development provides guidelines about fundamentals, a common framework, and a step toward developing a body of knowledge.

**A**s 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.<sup>1</sup> AOSD techniques' industrial maturity is evident in research<sup>2</sup> and in increasing interest in JBoss AOP (aspect-oriented programming, <http://labs.jboss.com/portal/jbossaop>), AspectJ ([www.eclipse.org/aspectj](http://www.eclipse.org/aspectj)), and Spring AOP ([www.springframework.org/docs/reference/aop.html](http://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 3D 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 3D approach throughout system analysis, development, and evolution.

In this article, we propose a model postgraduate curriculum for AOSD that does exactly

## Aspect-Oriented Software Development

The separation of concerns principle<sup>1,2</sup> suggests that it's best to treat each concern (that is, an interest that 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 "aspectization" improves system modularity, creating systems that are easier to develop, adapt, maintain, and evolve.<sup>3</sup> 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.<sup>4</sup> For more industrial applications, see [www.jboss.org](http://www.jboss.org), [www.eclipse.org/aspectj](http://www.eclipse.org/aspectj), and [www.springframework.org](http://www.springframework.org).

### References

1. D.L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules," *Comm. ACM*, vol. 15, no. 12, 1972, pp. 1053–1058.
2. E.W. Dijkstra, *A Discipline of Programming*, Prentice Hall, 1976.
3. A. Colyer and A. Clement, "Large-Scale AOSD for Middleware," *Proc. 3rd Int'l Conf. Aspect-Oriented Software Development*, ACM Press, 2004, pp. 56–65.
4. C. Tristram, "The Technology Review Ten: Untangling Code," *MIT Technology Rev.*, Jan. 2001; [www.ccs.neu.edu/research/demeter/aop/publicity/TR10\\_kiczales.html](http://www.ccs.neu.edu/research/demeter/aop/publicity/TR10_kiczales.html).

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). (Deliver-

ables and reports that describe many of our results and structures are available at [www.aosd-europe.net](http://www.aosd-europe.net)). Our 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](http://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 tailorable 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.

#### 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.<sup>1</sup>

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.<sup>1,3-6</sup> 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.<sup>1,7,8</sup> 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 imple-

mentation 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.<sup>9,10</sup> It surveys several semantic approaches, concentrating on one or two of them. 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<sup>11</sup> to aspects,<sup>12</sup> and relating static analysis to classes of temporal properties.<sup>9</sup> 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.<sup>13-15</sup> 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 as-

**The curriculum forgoes a rigid timetable and instead provides a structured body of knowledge tailorable to particular instantiation contexts.**

**Table 1****Curriculum instantiations and their relations to the model curriculum\***

Instantiation	Curriculum module					
	Module 1	Module 2	Module 3	Module 4	Module 5	Module 6
AOSD-Europe Summer School	No formal course—students did preparatory reading <sup>8</sup> on their own (0%)	Aspect-Oriented Design Early Aspects (28%)	Principles of AOP Languages (28%)	Formal Methods and Semantics for Aspects (23%)	Aspect-Oriented Middleware and a Case Study on Security Services (19%)	No formal course—the summer school treats AOSD in the context of OOP (2%)
Lancaster University	Introduction to AOSD Concepts (10%)	Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design (30%)	Aspect-Oriented Programming with AspectJ; AOP Models Beyond AspectJ; and Design Patterns Implementations and Assessment of AOP (30%)	Relating Aspectual Requirements to Proof Obligations (1%)	AOP Application in Persistent, Distributed Environments (20%)	Contrast AOSD with viewpoint- and use-case-based analysis; Comparatively assess OO and aspect-oriented realization of design patterns (9%)
EMOOSE (the European Master in Object-, Component-, and Aspect-Oriented Software Engineering)*	AOSD	Advanced Software Engineering Techniques; and Advanced Object-Oriented Software Architecture	Advanced Software-Engineering Techniques; Advanced Object-Oriented Software Architecture; and Object-Oriented Programming Languages and Their Implementation	Advanced Software Engineering Techniques	Distributed Object-Oriented Systems; and Advanced Object-Oriented Software Architecture	Object-Oriented Programming Languages and Their Implementation; Distributed Object-Oriented Systems; Advanced Object-Oriented Software Architecture; and Advanced Software Engineering Techniques
Trinity College Dublin	Introduction to AOSD Concepts (10%)	Aspect-Oriented Analysis and Design (25%)	Aspect-Oriented Programming with AspectJ; Introduction to AOP Paradigms Other than AspectJ (60%)	No course (0%)	No explicit course—students work in large groups to produce a distributed application (5%)	Object-Oriented Programming; Extreme Programming; Model-Driven Architecture (0%, or in group project)

\* Percentages represent classrooms hours spent on each module. 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 number of courses in EMOOSE ranges from 33 percent to 60 percent, depending on the individual projects' topics.

pect-oriented techniques, it's important to relate AOSD to the development paradigms, methodologies, and programming languages that you used to implement the underlying base application.<sup>16–18</sup> 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 in this section, 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 class-

room hours spent on each module in the program, where available.

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](http://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 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 ground the fundamental concepts in both 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,

**There's no one right way to implement the curriculum; each instantiation has specific needs.**

**Introducing AOSD within the concept of a full software module has worked well, giving students time to study, research, and appreciate the overall context.**

we risk students losing interest because of a lack of diverse learning activities. We tackle this issue by providing interactive sessions, such as the reading and roundtable sessions we mentioned earlier, which are the most popular feature of the course.

#### **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 comple-

ments 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.

Although 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 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 pri-

marily used AspectJ with Eclipse in the projects. 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 model-driven architecture 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 near 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 instantiation'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.

**W**ith 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

1. R.E. Filman et al., eds., *Aspect-Oriented Software Development*, Addison-Wesley, 2004.
2. A. Colyer and A. Clement, "Large-Scale AOSD for Middleware," *Proc. 3rd Int'l Conf. Aspect-Oriented Software Development*, ACM Press, 2004, pp. 56–65.
3. A. Rashid, A. Moreira, and J. Araújo, "Modularisation and Composition of Aspectual Requirements," *Proc. 2nd Int'l Conf. Aspect-Oriented Software Development*, ACM Press, 2003, pp. 11–20.
4. S. Clarke and E. Baniassad, *Aspect-Oriented Analysis and Design: The Theme Approach*, Addison-Wesley, 2005.
5. I. Jacobson and P.-W. Ng, *Aspect-Oriented Software Development with Use Cases*, Addison-Wesley, 2004.
6. D. Stein, S. Hanenberg, and R. Unland, "Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design," *Proc. 5th Int'l Conf. Aspect-Oriented Software Development*, ACM Press, 2006, pp. 15–26.
7. G. Kiczales et al., "Aspect-Oriented Programming," *Proc. 11th European Conf. Object-Oriented Programming*, LNCS 1241, Springer, 1997, pp. 220–242.
8. I. Aracic et al., "An Overview of CaesarJ," *Trans. Aspect-Oriented Software Development*, vol. 1, 2006, pp. 135–173.
9. S. Katz, "Aspect Categories and Classes of Temporal Properties," *Trans. Aspect-Oriented Software Development*, vol. 1, 2006, pp. 106–134.
10. R. Douence, O. Motelet, and M. Südholt, "A Formal Definition of Crosscuts," *Proc. 3rd Int'l Conf. Meta-level Architectures and Separation of Crosscutting Concerns*, LNCS 2192, Springer, 2001, pp. 170–186.
11. E.M. Clarke Jr., O. Grumberg, and D.A. Peled, *Model Checking*, MIT Press, 2000.
12. S. Krishnamurthi, K. Fisler, and M. Greenberg, "Verifying Aspect Advice Modularly," *Proc. 12th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.*, ACM Press, 2004, pp. 137–146.
13. A. Rashid and R. Chitryan, "Persistence as an Aspect," *Proc. 2nd Int'l Conf. Aspect-Oriented Software Development*, ACM Press, 2003, pp. 120–129.
14. S. Soares, E. Laureano, and P. Borba, "Implementing Distribution and Persistence Aspects with AspectJ," *Proc. 17th ACM SIGPLAN Conf. Object-Oriented Programming Systems, Languages, and Applications*, ACM Press, 2002, pp. 174–190.
15. J. Kienzle and R. Guerraoui, "AOP: Does it Make Sense? The Case of Concurrency and Failures," *Proc. 16th European Conf. Object-Oriented Programming*, LNCS 2374, Springer, 2002, pp. 37–61.
16. M. D'Hondt and V. Jonckers, "Hybrid Aspects for Weaving Object-Oriented Functionality and Rule-Based Knowledge," *Proc. 3rd Int'l Conf. Aspect-Oriented Software Development*, ACM Press, 2004, pp. 132–140.

**Within the network, structures are now being conceived that give AOSD-course graduates interesting future possibilities.**

**Table 2**

**Our curriculum's relationship to SWEBOX**

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

## About the Authors



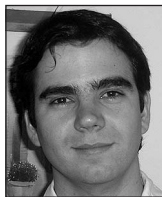
**Johan Brichau** is a postdoctoral researcher at the Université des Sciences et Technologies de Lille as part of the INRIA JACQUARD project at the Laboratoire d'Informatique Fondamentale de Lille. He's also a post-doctoral researcher at the Vrije Universiteit Brussel's Programming Technology Lab. His research interests include aspect-oriented programming language design, program querying and analysis using logic metaprogramming, and generative programming. He received his PhD in computer science from the Vrije Universiteit Brussel. Contact him at Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, France, LIFL-USTL Lille, Annexe Bat. M3-bureau 234, 59655 Villeneuve d'Ascq Cedex, France; 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 post-graduate diploma in computing from the Open University. Contact her at the Computing Dept., Infolab21, South Dr., Lancaster Univ., Lancaster LA1 4WA, UK; rouza@comp.lancs.ac.uk.



**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.

**Ellie D'Hondt** is a postdoctoral researcher in the Vrije Universiteit Brussel 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 science from Vrije Universiteit Brussel. She is active in the European Network of Excellence on Aspect-Oriented Software Development. Contact her at the Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussel, Belgium, eldhondt@vub.ac.be.

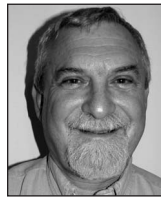
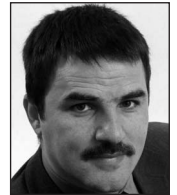


**Alessandro Garcia** is a computer science lecturer in Lancaster University's Computing Department. His research interests include empirical evaluation of AOSD, software architecture, exception handling, and agent-oriented software engineering. He received his PhD in computer science from the Pontifical 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. 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 the ACM. Contact him at Software Architecture Group, Hasso Plattner Inst. for Software Systems Eng., Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany; michael.haupt@hpi.uni-potsdam.de.



**Wouter Joosen** is a professor in the Katholieke Universiteit Leuven'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., The Technion, Haifa, 32000, Israel; katz@cs.technion.ac.il.

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 EMOOSE (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.Noyé@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.

Lancaster Univ., Lancaster LA1 4WA, UK; awais@comp.lancs.ac.uk.

**Mario Südholt** is a researcher at INRIA. He's also a lecturer at the École des Mines de 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.



- C. Zhang and H.-A. Jacobsen, "Quantifying Aspects in Middleware Platforms," *Proc. 2nd Int'l Conf. Aspect-Oriented Software Development*, ACM Press, 2003, pp. 130-139.
- J. Grundy, "Multi-Perspective Specification, Design, and Implementation of Software Components using As-

pects," *Int'l J. Software Eng. and Knowledge Eng.*, vol. 10, no. 6, 2000, pp. 713-734.

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