Int. J. Engng Ed. Vol. 21, No. 2, pp. 361–368, 2005
Printed in Great Britain.

0949-149X/91 $3.00+0.00
© 2005 TEMPUS Publications.

Introducing a Graduate Course on Aspect-Oriented Software Development*

BEDIR TEKINERDOĞAN
University of Twente, Department of Computer Science, P.O. Box 217, 7500 AE Enschede, The Netherlands. E-mail: bedir@cs.utwente.nl

Aspect-oriented software development (AOSD) is an advanced paradigm for separation of concerns (SOC) in software development, which provides explicit concepts to modularize so-called crosscutting concerns. After being accepted both by a broad community of researchers and the industry it is now getting introduced in courses in universities. This paper describes the experiences of the graduate course Aspect-Oriented Software Development that was introduced at Bilkent University in Ankara, Turkey. The lessons learned can be useful for peer educators who teach or aim to teach a similar course.

INTRODUCTION

ONE OF THE most important principles in software engineering for coping with complexity and achieving quality software is the separation of concerns principle [21]. This principle states that a given design problem involves different kinds of concerns, which should be identified and separated in different modules. The history of software development has experienced an evolution of different programming languages and design methods that have provided useful modularity mechanisms. However, as experienced in practice and generally acknowledged by researchers, it appears that these approaches are inherently unable to modularize all concerns of complex software systems. Some concerns like synchronization, recovery and logging tend to be more systemic, crosscut a broader set of modules and as such cannot be easily specified in single modules. This increases complexity and reduces several quality factors of software, such as adaptability, maintainability and reusability. Aspect-oriented software development (AOSD) is an advanced paradigm for separation of concerns (SOC), which provides explicit abstractions to modularize the crosscutting concerns and compose these with the system components [7].

AOSD has basically emerged from research in object-oriented programming. The problems encountered and the related solutions of aspect-oriented software development were originally discussed in various workshops in conferences such as ECOOP [6], OOPSLA [17] and ICSE [10]. The topics AOSD is concerned with are traditionally taught as part of object-oriented software development and software engineering courses.

After an increasing maturation and a consensus on the core AOSD concepts we can now speak of a separate AOSD community [1]. A separate conference called Aspect-Oriented Software Development is organized yearly, international projects such as the network of excellence on AOSD has been set up, and special issues of journals have started to publish papers on AOSD. The logical consequences of the separation and emergence of AOSD as an independent activity affect software engineering education as well. The need for separate courses on aspect-oriented software development is growing and an increasing number of courses on aspect-oriented software development are actually starting to be taught or are planned to be introduced into the curriculum.

This paper describes the introduction of the graduate course *Aspect-Oriented Software Development* at Bilkent University in Ankara, Turkey [25]. Several important lessons were learned from this course and we think that our experiences can be useful for peer educators who teach or aim to teach a similar course.

Our experiences may help guide educators on:

- planning an aspect-oriented software development course in a semester;
- dealing with an introductory course for which no suitable education material yet exists;
- adopting education forms that were used (presentations, project, demonstrations and workshop organizations);
- setting up a course project on AOSD;
- defining the evaluation criteria for the course;
- organizing a workshop within a course.

COURSE ORGANIZATION AND PLANNING

Existing software engineering courses at Bilkent university included courses on object-oriented programming, programming languages, software

* Accepted 26 October 2004.

architecture design, software engineering project management and object-oriented software development. The course was thus introduced as a graduate course, but selected undergraduate students were accepted as well. For these undergraduate students it was required that they fulfilled the following constraints:

- a fourth year student;
- a cumulative point grade average (CPGA) above or close to 3.0;
- having followed the course on Object-Oriented Software Development for which at least a B+ was required.

Setting these criteria for this relatively demanding course was necessary since there were too many students who did not really have a strong enough background to follow the course, but who were still eager to follow it. In this way, we actually also ensured that basically the best students in the department were selected for the course.

For organizing the course content and the planning over a 15-week semester we had first to decide on how to approach AOSD. An analysis of the historical developments of the contemporary AOSD approaches shows that some of these, such as Multi-Dimensional Separation of Concerns (MDSOC) [20, 24] and *DJ* [19] have evolved from program design principles based on separation of concerns, while others, such as Composition Filters (CF) [2] and AspectJ [11], have their roots in reflective computation [16]. The basic question was whether to explicitly focus on reflection or not. Although we think that reflection is important, we have chosen to build the course on the application of design principles rather than from reflective computation. The reason for this is twofold. Firstly, explaining reflection is very difficult and requires more background and conceptual effort. Secondly, current aspect-oriented programming approaches all seem to motivate the reason for AOSD based on design principles and reflection has become more implicit. This decision had a direct impact on both the content and the planning of the course; there was no stringent need to explain reflection in detail anymore. Rather we chose to only briefly explain it and referred to reflection papers in the recommendation list of papers of the course.

Directly related to design principles is the notion of design patterns [8] which represent reusable solutions to a set of problems within a given context. Design patterns are considered to represent the best possible application of design principles and as such can improve the modularity of the system. Before explaining aspects, an explicit discussion on design patterns would provide the student a more balanced view on dealing with design problems. Since design patterns have their limitations and cannot appropriately cope with crosscutting concerns, the classes on design patterns explicitly dealt with these problems so that the motivation for aspects became clear. Our aim in this course was to teach AOSD concepts so that they would be considered as complementary to existing practices such as the application of design principles and design patterns. As such we hoped to avoid 'aspect-hacking' in which every design problem would be mapped to aspects even when this could be elegantly solved by applying design principles or design patterns.

After the motivation for AOSD together with the application of design principles and design patterns was explained, we could start explaining the solutions provided by AOSD. We had deliberately chosen to select the name of the course as Aspect-Oriented *Software Development*, and not Aspect-Oriented *Programming* to provide a broader view. This decision resulted in the introduction of Aspect-Oriented Programming and Aspect-Oriented Design sections of the course.

For the Aspect-Oriented Programming part, we had to decide which aspect-programming languages to employ. Options for this were AspectJ, Composition Filters, HyperJ and DJ, all of which can be considered as prominent aspect-oriented programming languages [7].

In contrast to the programming part, the aspect-oriented design part was more challenging since there were very few publications that might be considered. Our aim with this part, however, was not to focus on a specific design approach but rather to teach that aspects occur also at the design phase and even at earlier phases.

Overall, we decided that the course had to be organized as follows:

- *General software engineering design principles.* Here we introduced the notion of software engineering [22], and the general design principles for achieving modular software. The separation of concerns principle was discussed in detail.
- *Object-oriented design patterns.* We used the patterns as explained in [8]. However, our aim was not to present a comprehensive overview of all the patterns but rather their context within the aspect-oriented software development paradigm.
- *Problems of crosscutting concerns.* This part showed that given the contemporary software development paradigms the application of design principles and patterns alone were not sufficient to cope with so-called crosscutting concerns, which cannot be localized in single modular units, are *tangled* with other concerns and *scattered* over different modules [13]. The problems of tangling and scattering and the reduction of modularity were illustrated using comprehensible examples.
- *General principles of AOSD.* After the problem and the motivation for AOSD were understood, the basic principles common to all AOSD approaches were explained. These include the notions of aspect, pointcut, joinpoint, advice, static crosscutting, and dynamic crosscutting [7].
- *Aspect-oriented programming.* Within this part the most prominent AOP approaches were

Table 1. Course content and form of evaluation

Week	Course Topic	Evaluation
1.	General overview of Course	
2.	Basic design principles	
3.	Object-Oriented Design Patterns	
4.	Object-Oriented Design Patterns	Written Exam (45 min) (covering Design Principles and OO Design Patterns)
5.	AspectJ	
6.	AspectJ **AspectJ Demo**	Written Exam (45 min) (covering AspectJ)
7.	Composition Filters (CFs)	
8.	Composition Filters **ComposeJ Demo**	Midterm Exam (90 min) (covering topics from week 1-week 8)
9.	Hyper/J *Start of Project Description*	
10.	Hyper/J **Hyper/J Demo**	
11.	Adaptive Programming and DJ	
12.	Aspect-Oriented Design (Notation)	
13.	Aspect-Oriented Design (Method)	
14.	Consultancy—No classes Individual group meetings with instructor	
15.	First Turkish Aspect-Oriented Software Development Workshop	Paper presentation, Workshop Paper, AspectJ Code
16.	Free (Exam week)	
17.	Final Exam	Final Exam (90 min) (covering topics from Wk 9–Wk 13)

presented (in order) AspectJ, Composition Filters, Hyper/J and DJ. A comparative analysis of the different approaches was also made.

- *Aspect-oriented design*. This part looked at very recent publications on both aspect-oriented modeling and the approach (method) for identifying aspects [3, 23, 26].

Table 1 shows how these topics were distributed throughout the 15-week semester, together with the evaluation schedule.

COURSE MATERIAL

Conventional courses on software engineering usually have no difficulty finding relevant textbooks, which include the important topics and suitable exercises. However, one of the problems with the introduction of relatively new topics into the curriculum is that suitable course material is hard to find. At the time of introducing the course (and now?) we could not find a textbook that we considered as being suitable and comprehensive enough.

We required a book that covered the state-of-the art AOSD techniques, and was not too specific dealing with, for example, a single Aspect-Oriented Programming language. Since we were unable to find such a book we decided to dispense with a textbook and rely on selected papers published by the Aspect-Oriented Software Development Community. The list of the papers that were used for the course is shown in Table 2. Note that in addition to the papers that were required for the course we have also included a list of recommended papers that were interesting but fall outside the scope of the current course.

COURSE IMPLEMENTATION AND PEDAGOGIC FORMS

Powerpoint presentations

The course met two days a week, for one one-hour and one two-hour lessons. Each lesson was presented using Powerpoint. As such more than 800 (animated) Powerpoint slides were prepared based on the guidelines for multimedia presentations [9], and they were effectively used to explain the topics. During the course, active discussions were encouraged as much as possible. In general, the students appreciated the use of electronic presentations especially when showing the conceptual problems such as tangling, scattering, and crosscutting. The presented course slides were put on the course home page (http://www.cs.bilkent.edu.tr/~bedir/ CS586-AOSD/) so that the students could download and view them later again when needed [11].

In-class demonstrations

The aspect-oriented software development course is an advanced course in software development and requires a considerably conceptual effort from students in order to understand both the problems and the proposed solutions. To clarify the topics and to make it more concrete, we focused on presenting as many examples as possible. In addition to the examples presented, we arranged demonstrations of three AOP languages, AspectJ, ComposeJ and HyperJ (downloaded from their corresponding websites). The demonstrations were shown after the discussion on the theory of the corresponding AOP approaches was completed, and were accompanied with separate presentations explaining the cases of

Table 2. Course material

General Software Engineering Principles/introduction to Aspect-Orientation
- B. Tekinerdoğan, On The Notion of Software Engineering, Chapter 2 in PhD thesis: Synthesis-based Software Architecture Design, University of Twente, Dept. of Computer Science, The Netherlands, 2000.
- W. Hursch and C. Lopes. *Separation of Concerns*, technical report, College of Computer Science, Northeastern University, 1995.
- G.Kiczales, J. Lamping, A.Mendhekar, C. Lopes, J. Loingtier, J. Irwin. *Aspect-Oriented Programming*, European Conference on Object-Oriented Programming (ECOOP), Springer-Verlag, LNCS 1241, June 1997.

Design Patterns
- E.Gamma, R.Helm, R. Johnson, J. Vlissides. *Design Patterns: Abstractions and Reuse of Object-Oriented Design*, European Conference on Object-Oriented Programming, Conference Proceedings, Springer-Verlag, Lecture Notes in Computer Science, 1993.

Aspect-Oriented Programming (AOP) approaches
- M. Aksit, L. Bergmans & B. Tekinerdoğan. *Aspect-Composition using Composition Filters*, in: Software Architectures and Component Technology: The State of the Art in Research and Practice, M. Aksit (Ed.), Kluwer Academic Publishers, pp. 357–382, 2001.
- H.Ossher & P. Tarr, *Multi-Dimensional Separation of Concerns in HyperSpace*, IBM-TJ Watson Research Center, Research report, RC 21452 (96717), Yorktown Heights, USA, 16 April 1999.
- P.Tarr, H.Ossher, W.Harrison & S.Sutton jr. *N Degrees of Separation: Multi-Dimensional Separation of Concerns*, in: Proc. International Conference on Software Engineering (ICSE 1999), 1999.
- Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. *An Overview of AspectJ*. In J. Lindskov Knudsen (ed.), ECOOP 2001 Object-Oriented Programming 15th European Conference, Budapest Hungary, pages 327–353. Volume 2072 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, June, 1997.
- K.Lieberherr, D. Orleans & J. Ovlinger. *Aspect-Oriented Programming with Adaptive Methods.*
- D. Orleans & K. Lieberherr. *DJ: Dynamic Adaptive Programming in Java.*

Aspect-Oriented Design (Notation)
- S. Clarke, W. Harrison, H. Ossher, P. Tarr. *Subject-Oriented Design: Towards Improved Alignment of Requirements, Design, and Code*. Proceedings of OOPSLA '99.
- S. Clarke, R.J. Walker. *Composition Patterns: An Approach to Designing Reusable Aspects*. In proceedings of the 23rd International Conference on Software Engineering (ICSE), Toronto, Canada, May 2001.
- D. Stein, S. Hanenberg, & R. Unland. *A UML-based Aspect-Oriented Design Notation for AspectJ*, in: G. Kiczales (ed.), Proc. of First ACM International Conference on Aspect-Oriented Software Development, Enschede, The Netherlands, April, 2001.

Aspect-Oriented Design (Process)
- B. Tekinerdoğan, M. Aksit. Separation and Composition of Concerns through Synthesis-Based Design, ACM OOPSLA'2000 workshop on Advanced Separation of Concerns, Minneapolis, October 2000.
- B. Tekinerdoğan and M. Aksit. Deriving Design Aspects from Canonical Models, in Object-Oriented Technology, S. Demeyer and J. Bosch (Eds.), LNCS 1543, ECOOP'98 Workshop Reader, Springer Verlag, pp. 410–413, July 1998.

Other recommended reading (not part of the syllabus)
- L. Bergmans, *Composition Filters Model*, chapter PhD thesis
- P. Maes, *Concepts and Experiments in Computational Reflection*, ACM SIGPLAN Notices, v.22 n.12, p.147–155, Dec, 1987.
- D. Parnas. *On the Criteria To Be Used in Decomposing Systems into Modules*, Communications of the ACM, Vol. 15, No. 12, pp. 1053–1058, December 1972.

the demonstrations. The demonstrations included both several predefined scenarios and new scenarios that came up after discussions during the lesson. In addition to the plenary demonstrations, the students were also given the opportunity to redo the demonstration at home. In our experience these demonstrations have not only been of great help for an improved understanding of the presented topics but also improved the enthusiasm for the course topics. In fact, with the demonstrations the students observed for the first time the problems that AOSD addresses, the approach, and how aspects were implemented, compiled, and debugged.

Homework

The students were given homework on the topics that were discussed in the corresponding week. These homework assignments were not graded. The homework included small assignments on aspect-oriented design and aspect-oriented programming. The assignments were shortly discussed in the course and, if needed, explained in more detail afterwards. One of the important homework assignments was to download and install the prominent aspect-oriented tools for AspectJ, HyperJ and ComposeJ. The students had to experiment with the examples that were implemented in the tools. These were, for example, the design of a space war game and a concurrent mail delivery system. By doing this, the students got practical experience on using these important tools in particular, and the obstacles in aspect-oriented programming in general. Further, they were also able to define their preferences for the tool and aspect-oriented programming language that they would use later on in the project assignments.

Project

The course included one midterm project in which the basic aspect-oriented concepts and techniques were practiced. The project's aim was to clarify the concepts that had been taught during the classes using teamwork [14]. For this, the students formed groups of four, working together on a case that they had selected themselves. There

Table 3. Examinations and grading

Task	Description	Weight
Written (Short) Exam	During the course—45 min.	15%
Project	Report (75%) Presentation (15%) Code (10%)	35%
Midterm Exam	Open book 120 mins.	25%
Workshop Paper	Write a workshop paper based on project results	10%
Final Exam	Open book 120 mins.	15%

were 36 students in the class, which led to 9 groups. The cases they selected had to be the design of a sufficiently complex and relevant application. To illustrate the diversity of the topics we list the cases that were selected:

- Applying AOSD to Legacy Systems
- Aspect-Oriented Design of Ant Simulation
- Aspect-Oriented Testbed for Cost-Sensitive Classification Algorithms
- Identifying Domain Specific Aspects in a Pathway Analysis Toolkit
- A Development Toolkit for Aspect-Oriented Programming
- Aspect-Oriented UML Class Diagram Drawing Tool
- Aspectual Development of Concurrent Versioning System
- Aspect-Oriented Design for a Peer-to-Peer File Sharing System
- Aspect-Oriented Worker Thread Pool Framework.

Most of these cases were existing cases or research topics of the group in which the students had to finalize their M.Sc. assignments. As such the projects had also an indirect positive effect since the students directly applied the concepts in the real research setting.

Another important aim of the project was to learn how to identify crosscutting concerns (problems) and specify, implement and evaluate aspects (solutions). For this, the students had to formulate the problem of crosscutting concerns in the given case, plan the solution, design and translate the solution, and finally implement the solution. As such we aimed to support not only the understanding of common domain-specific issues of AOSD but also the individual problem-solving skills as described in for example [4].

To exercise the AO*P* problems, the selected cases had to include at least two development aspects and three production aspects [11]. *Development aspects* represent the aspects that are used to facilitate the development of aspects, such as tracing, debugging and testing aspects. *Production aspects* represent aspects that are inherently intended to be included in production. Considering multiple aspects at a time is important to highlight the dependencies between aspects, and the additional complexities they impose on the eventual design. In general, the development aspects of the groups did not differ much because there seems to be a fixed set of these aspects. The selected production aspects on the other hand were all different because every group had a different case.

Using the selected cases the following subtasks had to be performed:

- Object-oriented design of the case: to include at least two object-oriented design patterns.
- Develop change scenarios which cannot be easily integrated into the case.
- Problem Statement: explain the shortcomings of the object-oriented model with respect to the crosscutting concerns and show this using the defined change scenarios.
- Aspect-Oriented Design of the case using a selected notation.
- Aspect-Oriented Programming in AspectJ: implement 5 aspects (3 production aspects, and 2 development aspects).
- Provide alternative aspect specifications using Composition Filters, HyperJ or DJ.
- Evaluate the aspect-oriented design and a compare the chosen AOP approaches.

Note that for the implementation of aspects we have chosen to use AspectJ. The reason for this was that AspectJ had the most mature integrated development environment and related documentation.

The deliverables of the project included a report (about 40–50 pages), an aspect-oriented program in AspectJ, a PowerPoint presentation which was presented in class, and a workshop paper. The students were in general very positive about the project since it provided them the opportunity to experience a real aspect-oriented software development process.

ASSIGNMENT AND GRADING

The final course grade was based on a set of written exams/assignments, which were used to evaluate students' understanding of the course material. These are listed in Table 3. There were two short examinations of 45 minutes each, and one midterm and one final exam of 120 minutes. The short examinations helped to assess the student's knowledge of particular topics. The midterm and final exams were more comprehensive and focused on a broader understanding and

application of the topics. The examinations did not only help to evaluate and grade the students but also ensured indirectly that the course topics were studied on time. This was important considering the relative advanced nature of the course. Besides the examinations, which were evaluated individually, the students also got a group grade for the project and the related workshop (see next section). Note that the project counts for a significant part of the grade, which emphasizes its importance.

WORKSHOP ORGANIZATION

During the project, complex cases were selected from industry and ongoing projects at the university and these were analyzed for aspects and re-engineered as aspect-oriented designs. Aspect-oriented programs have been implemented in AspectJ and a comparison made with other prominent AOP approaches such as Composition Filters, Hyper/J and DJ. This resulted in a unique collection of valuable aspect-oriented designs, which show the strengths, and weaknesses of AOSD. To make these valuable practices public for a broad audience we decided to organize the First Turkish Aspect-Oriented Software Development Workshop (http://www.cs.bilkent.edu.tr/taosd03/) for which we would invite participants from industry and other universities. This was a unique experience since most of the students had never participated in a workshop before. In addition, since the topic was introduced for the first time in Turkey, the audience would consist of participants who were new to this paradigm. Consequently, we thought that organizing such a workshop would be beneficial for both our students and external participants. In particular, with the organization of the workshop we had the following goals in mind:

- Trigger academic and industrial activities in the AOSD domain in Turkey.
- Show real-world example cases using different AOSD approaches to highlight the current state of the art in the AOSD community.
- Show the lessons learned from aspect-oriented software development.
- Share our ideas with respect to aspect-oriented software development education on AOSD in Turkey.
- Trigger new research topics on AOSD.

The project results have been presented as workshop papers. Because most students in the AOSD course wrote a workshop paper for the first time, they also received a short course on how to write workshop papers and in addition got extensive feedback on their papers by the instructor. The papers present a broad range of aspects in different applications, discuss the crosscutting problems and as such provide a unique set of aspect examples. In addition, the papers also highlight the obstacles in AOSD and provide some triggers for new research directions. As such, we consider the papers in the published workshop proceedings as both useful for novice aspect-oriented software developers and aspect-oriented researchers.

Based on the number of the project groups we had in total nine presentations, each of them took 30 minutes including a 10-minute plenary discussion after each presentation. There were three presentation sessions, and each presentation included one real demo showing sample aspect code in AspectJ. The workshop was concluded with a plenary session in which we summarized the results and discussed our future plans to distribute and share our knowledge on AOSD in Turkey.

The workshop was really successful; about 70 participants both from industry and academia registered for it. We hope that by organizing such a workshop we have contributed to the early discovery and active involvement of aspect-orientation in the Turkish software engineering community.

CONCLUSION

In this paper we have described the introduction of the graduate Aspect-Oriented Software Development course at Bilkent University in Ankara, Turkey. The course organization has provided a number of lessons that might be useful for peer educators who teach or plan to teach a similar course.

Since no suitable textbooks for teaching AOSD exist yet, the course material was collected from a set of important AOSD research papers. In our experience we have seen that this does not form a real obstacle. In fact we could even say that this helped to improve scientific reading and writing skills, which is essential for graduate students. For this reason, even if an appropriate textbook were available we would still consider including some research papers in the course material.

Regarding the historical development of AOSD and the motivation for adopting AOSD we have chosen to start the course by explaining fundamental software design principles. This was followed by design patterns which can be considered as the optimal approaches in which the design principles are applied. After having explained that some concerns might still be crosscutting and as such reduce modularity of software, we have first taught Aspect-Oriented Programming and then Aspect-Oriented Design.

The use of examples and real demonstrations of prominent AOP approaches in class have helped to increase the understanding of AOSD topics and the enthusiasm of the students for the course topics. The project helped students gain a concrete understanding of the aspect-oriented concepts by designing and implementing a real example case.

Both the problems and the solutions afforded by AOSD were explicitly covered in the project.

The project results formed a unique collection of AOSD design examples, which we planned to make public for a broader audience by organizing a workshop. The organization of a workshop within a course was really a unique experience and helped increase not only the scientific reading, writing and presentation skills of the students but also supported the popularization of the AOSD concepts in Turkey.

The course was evaluated by the students and the results were overly positive. On a scale from 1 (low) to 5 (high), the overall effectiveness of the course was rated as 4.71, which is considered an exceptionally good evaluation.

In the future we plan to give the same course in a similar way. The only extension might be to include laboratory work [18] but we do not consider this really necessary because in the current organization of the course the students already do considerable practical work.

Acknowledgment—I would like to thank all the CS586 students for their enthusiasm and their hard work during this course. Finally, I would like to thank David Davenport at Bilkent University for reviewing earlier versions of this paper.

REFERENCES

1. AOSD community home page. http://aosd.net
2. M. Aksit, L. Bergmans and B. Tekinerdogan, Aspect-composition using composition filters, in *Software Architectures and Component Technology: The State of the Art in Research and Practice*, M. Aksit (ed.), Kluwer (2001) pp. 357–382.
3. S. Clarke, R. J. Walker, Composition patterns: an approach to designing reusable aspects*, Proc. 23rd Int. Conf. Software Engineering (ICSE)*, Toronto, Canada, May 2001.
4. F. Deek, M. Turoff and J. McHugh, A common model for problem solving and program development, *J. IEEE Trans. Educ.*, **42**(4), November 1999, pp. 331–336.
5. E. W. Dijkstra, The structure of 'THE' multiprogramming system, *Communications of the ACM*, **11**(5), 1968, pp. 341–346.
6. European Conference on Object-Oriented Programming (ECOOP). http://www.ecoop.org/
7. T. Elrad, R. Fillman and A. Bader, Aspect-oriented programming, *Communication of the ACM*, **44**(10), October 2001.
8. E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design patterns: abstractions and reuse of object-oriented design, *Proc. European Conf. Object-Oriented Programming*, Springer-Verlag, Lecture Notes in Computer Science, 1993.
9. F. T. Hofstetter, *Multimedia Presentation Technology*, Belmont, California: Wadsworth Publishing Company (1993).
10. Int. Conf. Software Engineering. www.icse-conferences.org
11. S. Kanya, Online education expands and evolves, *IEEE Spectrum*, **40**(5), May 2003, pp. 49–51.
12. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm and W. G. Griswold, An overview of AspectJ, in J. Lindskov Knudsen (ed.), *ECOOP 2001 Object-Oriented Programming 15th European Conf.*, Budapest Hungary, Volume 2072 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, June, 1997, pp. 327–353.
13. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier and J. Irwin, Aspect-oriented programming, *Proc. ECOOP '97*, Springer-Verlag LNCS 1241.
14. M. Krishnan, S. Das and S. A. Yost, Team-oriented, project-based instruction in a new mechatronics course, *Frontiers in Education Conf.*, 1999.
15. K. Lieberherr, D. Orleans and J. Ovlinger, Aspect-oriented programming with adaptive methods, *Communications of the ACM*, **44**(10), October 2001, pp. 39–43.
16. P. Maes, Concepts and experiments in computational reflection, *ACM SIGPLAN Notices*, **22**(12), Dec, 1987, pp. 147–155.
17. Object-Oriented Programming, Systems, Languages and Applications Conference. http://www.oopsla.org/.
18. L. Ohlsson and C. Johansson, A practice driven approach to software engineering education, *IEEE Trans. Educ.*, **38**(3) August 1995.
19. D. Orleans and K. Lieberherr, DJ: Dynamic Adaptive Programming in Java, in *Reflection 2001: Meta-level Architectures and Separation of Crosscutting Concerns*, Springer Verlag (2001)
20. H. Ossher and P. Tarr, *Multi-Dimensional Separation of Concerns using Hyperspaces*, IBM Research Report 21452 (1999).
21. D. Parnas, On the criteria to be used in decomposing systems into modules, *Communications ACM*, **15**(12), Dec. 1972, pp. 1053–1058.
22. R. S. Pressman, *Software Engineering: A Practitioner's Approach*, Mc-Graw-Hill (1994).
23. D. Stein, S. Hanenberg and R. Unland, A UML-based aspect-oriented design notation for AspectJ, *Proc. First ACM Int. Conf. Aspect-Oriented Software Development*, Enschede, The Netherlands, April, 2001.
24. P. Tarr, H. Ossher, W. Harrison and S. M. Sutton, Jr., N degrees of separation: multi-dimensional separation of concerns, *Proc. ICSE 21*, May, 1999.
25. B. Tekinerdogan, CS586-Aspect-Oriented Software Development Course Home Page, Bilkent University, Ankara, Turkey (2003). www.cs.bilkent.edu.tr/~bedir/CS586-AOSD/
26. B.Tekinerdoğan and M. Akşit, Deriving design aspects from conceptual models, Demeyer, S. and Bosch, J. (eds.), *Object-Oriented Technology, ECOOP '98 Workshop Reader, LNCS 1543*, Springer-Verlag, pp. 410–414 (1999).

Bedir Tekinerdoğan received his M.Sc. degree in Computer Science in 1994, and a Ph.D. degree in Computer Science in 2000, both from the University of Twente, The Netherlands. From September 2000 until August 2002 he fulfilled a post-doc position at the University of Twente. From September 2002 until July 2003 he was an assistant professor at Bilkent University in Turkey. Currently he is an assistant professor at the University of Twente in The Netherlands. His basic research topics are aspect-oriented software development, software architecture design, and software product line engineering. He has served on the program committee and organising committee for different conferences and workshops related to aspect-oriented software development, software architecture design, and object-oriented software development. From 1994 on, he has taught courses on object-oriented programming, object-oriented analysis and design, object-oriented design patterns, software architecture design and aspect-oriented software development.