

# Authorization Enforcement Usability Case Study

Steffen Bartsch

Technologie-Zentrum Informatik TZI,  
Universität Bremen, Bibliothekstr. 1, 28359 Bremen, Germany  
[sbartsch@tzi.org](mailto:sbartsch@tzi.org)

**Abstract.** Authorization is a key aspect in secure software development of multi-user applications. Authorization is often enforced in the program code with enforcement statements. Since authorization is present in numerous places, defects in the enforcement are difficult to discover. One approach to this challenge is to improve the developer usability with regard to authorization. We analyze how software development is affected by authorization in a real-world case study and particularly focus on the loose-coupling properties of authorization frameworks that separate authorization policy from enforcement. We show that authorization is a significant aspect in software development and that the effort can be reduced through appropriate authorization frameworks. Lastly, we formulate advice on the design of enforcement APIs.

## 1 Introduction

Access control and, more specifically, authorization is a prevalent security requirement for multi-user applications, particularly for Web applications with the well-known advantage of efficiently serving large numbers of users. Research on authorization is focused on authorization models and algorithms for the enforcement of policies [1, 5, 12, 33]. Only a fraction of the publications on authorization measures takes usability aspects into account, although it is widely accepted that a lack of security usability leads to vulnerabilities [32]. In case of authorization, the vulnerabilities may for example stem from end users circumventing imposed authorization measures to efficiently complete the work at hand [19].

Authorization usability can be categorized according to three perspectives: (1) the functional perspective of end users who may be hindered by inappropriate restrictions, (2) the security perspective of policy authors who elicit and codify authorization requirements into a policy, and (3) the development perspective of software developers who implement authorization enforcement, for example by placing adequate enforcement statements in the source code. This work primarily focuses on the development perspective of enforcement usability. Our main target is the

effectiveness, that is the precise implementation of enforcement, and the efficiency, affected by the effort necessary to achieve the effectiveness. Developer usability is also concerned with the security perspective since the policy is often specified concurrently to development in case of custom software development. This development model is particularly widespread for agile development of Web applications.

Enforcement usability is an important aspect of secure software development, since not adequately enforcing authorization leads to security risks [18]. A challenge is that, as a cross-cutting concern, authorization is often enforced on throughout the program. Consequently, authorization is difficult to test automatically in its entirety so that many authorization-related defects are only found in manual acceptance tests. A negative example of the usability are repetitive and complex enforcement statements because redundancy may lead to defects when developers implement evolutionary changes that are imminent in software development [22]. Interestingly, there has been little systematic work on authorization enforcement usability, despite the prevalent authorization requirements. To the best of our knowledge, there has been no prior work with a focus on how authorization is developed and the interplay of policy and enforcement changes in the development of custom applications.

This paper pursues to improve the understanding of how authorization affects software development. We assess the impact of authorization on software development in a case study of a business Web application. In particular, we study how the loose coupling of policy and enforcement, a well-known separation of concerns design principle [11], affects real world software development in this case study. The approach is an in-depth analysis of authorization in a real-world agile development project that employs a widely-used authorization framework. We evaluate the project's source code repository commits. Before describing the case study, we give a brief background on authorization usability and the framework that is used in the project. Following the study results, we offer advice on enforcement API design based on experiences from the case study.

## 2 Authorization Usability in Software Development

There are numerous authorization models, ranging from simple discretionary access control, such as access control lists, to logic-based models [27]. For the relevant authorization aspects in this paper, policy specification and authorization enforcement, each model has a specific form, differing in usability for policy creators and software developers.

## 2.1 Policy Specification

With many different authorization models and associated policy specification languages, the usability challenges naturally differ. For privacy policies, Reeder et al. identified five usability challenges: policy creators need to be able to efficiently group objects, use consistent terminology, understand the default rules, uphold the policy structure and solve rule conflicts [25]. In case of role-based policies, Brostoff et al. found that the primary challenges for policy creators lie in understanding the policy structure and the overall authorization paradigm [7].

Languages for authorization policy specification are usually defined as domain-specific languages (DSL) [10]. For the usability of an authorization language, the distance between humans and the syntax is important [24]. Inglesant et al. studied how security experts formulated policies using a controlled natural language with an explicitly reduced distance. They also proposed an iterative process for policy specification, which is necessary for users to understand the authorization model [17]. Similarly, Stepien et al. suggest an intermediate natural language notation to simplify the creation of XACML policies [29]. Apart from policy languages, tool support for the creation of policies has been studied. Zurko et al. developed a policy editor for policies similar to RBAC (Role-based Access Control) based on usability testing and user-centered design that allowed novice users to produce meaningful results in under one hour [34]. Herzog and Shahmehri studied the usability of the Java policy tool and observed that help on semantics was missing. They also concluded that the tool is inadequate for expert users and promotes lax policies [16]. Lastly, security policy management does not only encompass the definition, but also decision-making regarding the policy. For this purpose, Rees et al. propose a framework for the development of high-level security policies [26].

## 2.2 Authorization Enforcement

Systematic authorization enforcement is not a new concept and has already been described in form of reference monitors in the 1970s [2]. Current authorization enforcement mechanisms come in a variety of approaches, often applied in combinations [23]. Operating system-based enforcement relies on operating system mechanisms that check permissions on the level of files or processes [15]. In runtime system-based mechanisms, the running program is encapsulated and every call to a protected component is first checked against a reference monitor; an example is

the Java security model [14]. For language-based approaches, a compiler generates additional enforcement code for the specified authorization policy [13]. Similarly, aspect-oriented programming (AOP) can be employed to enforce authorization at the join points [20]. A related approach is the transformation of program bytecode to enforce program behavior, for example in Java programs to secure hosts against potentially malicious mobile code [23]. In these approaches, the program code is regarded as potentially hostile, requiring external supervision. In contrast, many authorization decisions are made in program code for external users, permitting a cooperative approach of program code and authorization. In these API-based approaches, enforcement hooks are placed into the source code, for example as `checkPermission` calls in Java [14] or as security hooks in the Linux kernel [18].

The challenges in authorization enforcement are the correct placement of enforcement statements. Errors in placement were for example found in Linux kernel modules [18]. Measures to enforce authorization as part of the architecture may be circumvented by accident through unanticipated control flow [28]. It is particularly difficult to achieve adequate authorization enforcement in software evolution, which often requires program comprehension for further development [31]. Frequently, information on why a certain state is reached is missing in development [21]. The API design also affects the enforcement comprehensibility [9, 30]. Therefore, enforcement was ideally implemented once and could be left unmodified even when authorization requirements change [6]. Vice versa, the authorization policy would not need to be modified even with functional changes as long as the authorization requirements remain unchanged. Loose coupling of authorization policy and enforcement is needed as the separation of concerns [11].

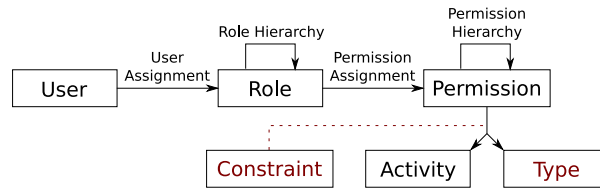
### 3 Authorization Framework: `declarative_authorization`

Authorization in the case study is based on `declarative_authorization`<sup>1</sup>, a widely-used open source authorization framework for the Web development platform Ruby on Rails [4]. It aims particularly at offering loose coupling of authorization policy and enforcement. Moreover, the framework supports secure software development with readable and maintainable authorization policies and enforcement statements.

The authorization model, depicted in Figure 1, is similar to classical RBAC [12] in that roles are assigned to users and permissions to roles

---

<sup>1</sup> [http://github.com/stffn/declarative\\_authorization](http://github.com/stffn/declarative_authorization)



**Fig. 1.** `declarative_authorization` authorization model

with role and permission hierarchies. Instead of composing permissions of object and activity, the framework combines object types (contexts) and activities with constraints to increase the maintainability. Constraints are either defined on attributes of an object of the type, possibly spanning multiple nested object relations, or according to specific permissions on related objects. The policy language, although based on Ruby syntax, is aimed to be a human-readable and intuitively comprehensible textual DSL. A change management tool supports the modification by non-experts [3].

Authorization enforcement follows the typical model–view–controller (MVC)-approach that Ruby on Rails encourages, providing enforcement on multiple layers. Primarily, enforcement occurs on the model (object-relational mapping) and controller (business logic) layers. Declarative enforcement statements limit permissions on create, read, update, delete (CRUD) activities on the model and respective access through HTTP requests on controller actions. Additionally, the authorization policy is enforced through database query rewriting.

## 4 Authorization Development Case Study

To gain further insights on authorization in software development, we studied the development of a custom business Web application for a medium enterprise from the automotive supplier industries. At the time of examination, development had been ongoing for 2.5 years with small teams that employed agile development practices. The application had been in full productive use for 1.5 years.

As indicated in the introduction, we pursue the following research goals in this case study: First, what is the actual impact of authorization on agile software development? The hypothesis is that agile development causes frequent changes of functional requirements, including authorization requirements. Thus, a substantial part of the software development

should affect authorization (H1). Second, how does a loose-coupling authorization framework help in authorization development? We expect that changes to the requirements often only affect either the policy or the enforcement (H2). Third, in which cases does the separation of concerns not work? Despite loose coupling, we still expect that there are authorization enforcement modifications for authorization requirement changes and vice versa (H3).

#### 4.1 Methodology

For the study, we analyzed the commits to the development branch of the subversion repository over almost 1.5 years beginning with the introduction of the `declarative_authorization` plugin in the project, which superseded a very simple authorization mechanism. We target the efficiency aspect of developer usability and, thus, would need to study the effort that developers spend on different tasks. Since development effort is difficult to assess directly, our analysis is based on commit counts and the relation between the commits that touch authorization development aspects. The rationale is that a developer generally risks to introduce defects and spends effort if a commit affects authorization, impacting the effectiveness and efficiency, respectively. We focused on two distinct aspects, policy changes and changes to authorization enforcement, and manually coded commits according to the reasons of modifications. Since a commit may contain several changes, all coding was non-exclusive.

Firstly, we determined the total number of commits on the development branch. For policy changes, we analyzed commits to the authorization rules configuration file and coded the commits into the categories that are listed in Table 1 using the commit log entries, inspection of the version differences and developer knowledge of underlying change reasons.

For enforcement, we automatically analyzed the differences between versions and considered a commit enforcement-related if the changeset

**Table 1.** Categories of reasons for authorization policy changes

Refactoring	Changes to authorization policy for cosmetic and readability reasons without affecting the authorization test results
Bugfix	Authorization policy modification to fix bugs caused by the policy
Authorization requirement	Modifications related to changed authorization requirements without further application functionality changes.
Functionality changes	Authorization policy changes that are caused by functional changes to the application

**Table 2.** Categories of reasons for authorization enforcement changes

Bugfix	Bugfixes in application code that relate to enforcement
Requirement changes	Changes in enforcement that result from changed authorization requirements
Refactoring	General application refactoring that affects enforcement
Authorization-related Refactoring	Refactoring in application code to change authorization refactoring
Functional changes	Functional changes in application source code that affect enforcement

contained a change line with an enforcement statement. All enforcement commits were manually verified and categorized according to Table 2. For a more detailed analysis of enforcement commits, we also coded the commits according to the modification type, either addition, modification or removal.

## 4.2 Results

*Reasons for changes* The results from the commit analysis are shown in Table 3. For each commit type, a line lists the quantity and, if applicable, the ratio against the total commit quantity and against the enforcement or policy-related commits. In addition, the raw categories are grouped in multiple cases as described below. Commits may fall in multiple categories, so that ratios do not sum up to 100%.

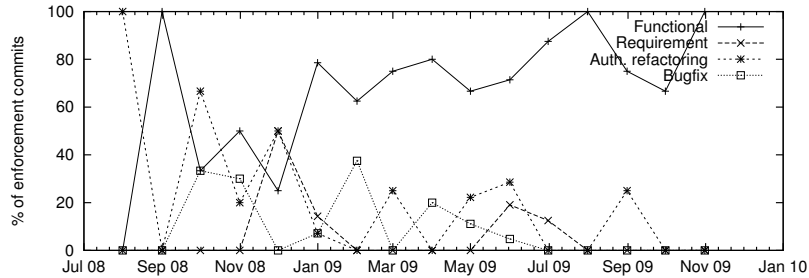
From the quantitative results in Table 3, we deduce several findings. Interestingly, almost a quarter (23%) of all commits are related to authorization, almost 10% to the policy. Looking at the impact of authorization requirement changes, the results show that 7% of all commits (76% of all policy-related commits) result from changed authorization requirements, including requirements from functionality changes. This result shows that there is a high number of authorization requirement changes as expected from agile development. In 67% of the commits relating to the policy, authorization behavior is changed without functionality changes, that is only authorization bugfix or requirements changes. Moreover, there are 33 commits caused by authorization requirement changes on the policy compared to only 9 for enforcement, preventing unnecessary code changes in one quarter of the changes. Only 8% of the enforcement change commits are related to authorization requirement changes without further functionality changes, so it again mostly seems to suffice to change the policy. These results suggest a loose coupling of policy and enforcement.

**Table 3.** Authorization commit reasons quantities

	# commits	of total commits	of enforcement/ policy commits
<b>Total commits</b>	610		
Total authorization-related	137	22.5 %	
<b>Policy-related</b>	58	9.5 %	
Functionality	17	2.8 %	29.3 %
Bugfixing	8	1.3 %	13.8 %
Policy refactoring	15	2.5 %	25.9 %
Authorization requirements	33	5.4 %	56.9 %
Authorization req.-related	44	7.2 %	75.9 %
Non-functional	39	6.4 %	67.2 %
<b>Enforcement-related</b>	116	19.0 %	
Functionality	52	8.5 %	44.8 %
General refactoring	35	5.7 %	30.2 %
Authorization bugfixes	11	1.8 %	9.5 %
Authorization refactoring	20	3.3 %	17.2 %
Authorization requirements	9	1.5 %	7.8 %
Authorization-only	37	6.1 %	31.9 %
Non-authorization	83	13.6 %	71.6 %

On further analyzing the enforcement-related commits, the results show that 72% of all enforcement-related changes are for general refactoring or functional changes. This result is important as it indicates that most enforcement changes are really necessary because the changes are part of the functional application development. While 9% of all commits involve enforcement changes that are caused by functionality changes, functionality changes cause only policy changes in 3% of all commits. This suggests that the loose coupling of policy and enforcement also works in the reverse direction: functional changes are more likely to affect the enforcement than the policy.

*Enforcement modifications* Although, ideally, no authorization-related modifications of enforcement should be necessary, one third (32%) of the enforcement changes relate to authorization bugfix, refactoring and requirement changes. We analyzed the reasons of authorization enforcement changes further in two ways: by examining the distribution of commits over time and by coding the commits by enforcement modification types. The distribution of enforcement commits over time relative to the total number of enforcement commits are shown in Figure 2. In the diagram, the most striking development is that authorization-related enforcement commits are decreasing after a transitional period of five months. Most of the authorization refactoring occurred in the transitional period. Later,



**Fig. 2.** Distribution of reasons for enforcement commits over time

the authorization enforcement stabilizes with only little refactoring necessary.

Further insights may be gained from the types of enforcement modifications, shown as a matrix for modification operations and reasons in Table 4. In this project, the addition and modification changes for authorization bugfixing can be primarily attributed to originally missing or wrong permission checks, for example found in acceptance tests. Similarly, addition enforcement commits for authorization requirements were caused by missing checks that were needed to enforce a requirement. This is also the reason for the modification commits, since the framework’s enforcement statements either take a general context (object type) or specific object as basis for the decision. This may need to be modified for some authorization requirement changes.

### 4.3 Discussion

The findings are in line with the hypotheses. We observed that a significant part of the development was related to authorization. As we expected, agile development causes frequent changes to functional and authorization requirements over time (H1). Functionality-related autho-

**Table 4.** Enforcement-related commit quantities by modification reason and type

	Addition	Modification	Removal	Total
Functionality	29	26	5	52
General refactoring	4	31	3	35
Authorization bugfixes	7	5	0	11
Authorization refactoring	10	13	2	20
Authorization requirements	7	5	0	9
Total	49	67	10	

rization changes affected enforcement to a higher degree. Similarly, policy modifications are more often caused by authorization requirement changes. Thus, we found a loose coupling between policy and enforcement that improves the developer usability by reducing effort and preventing errors (H2). We still observed authorization requirements that affect enforcement and vice versa. The primary reasons were a large amount of transitional work in the first months after the framework introduction. To a lesser extent, enforcement additions and modifications were also necessary to fulfill changing authorization requirements (H3).

Since we conducted a single-project case study, we acknowledge several threats to the study's validity. For the scope validity, the results are mainly applicable to custom-developed Web applications that involve a non-trivial amount of authorization, preventing the definition of the entire authorization requirements in advance. The specific development process is also important, since the agile methods in this project caused continuously changing functional and authorization requirements. As most plan-based development processes also have changing requirements [22], we suppose that the effects will be similar in other process models. It would be interesting to investigate the influence of the process model in future research. The development context could also pose a threat to the validity since the development was conducted in the university realm. However, development was very focused on the product owner's requirements and was conducted by skilled developers of at least two years of experience with the development framework. We found no significant effects from trainings and no developer took security courses while on the project. Lastly, the specific programming language, Web development platform and authorization framework may have affected the results. We analyzed whether changes to the framework resulted in additional refactoring but the API was rather stable and we could only identify one commit in which non-backward compatible changes to the framework required refactoring in the application code. There was no authorization refactoring from newly introduced framework functionality. Still, we will study other authorization frameworks and platforms in future research.

Other threats to validity are posed by the study author's involvement in the development. While it is common in the humanities to conduct empirical work with active involvement of the study authors, for instance in action research [8], it is less common in computer science. Still, we believe that the involvement has not significantly influenced the results as the analysis was fully conducted *post-mortem* and, more importantly, the analysis was only decided after the studied period was over. Thus, the

author should not have been biased in development. Moreover, developer involvement in the study was inevitable as the coding of commits was only possible with knowledge of the system and the development process. The primary source for a potential author bias in the study is the coding of commits into the categories. Some of the categories have fluent boundaries so that the threat of a biased coding is present, particularly with only one person having coded the commits. On the other hand, we were aware of this threat and separated the coding from the analysis phase to reduce the potential effect. While we cannot rule out the effect completely, we are confident that it did not significantly affect the results.

Lastly, threats to validity originate from the methodology of analyzing commit quantities instead of the actual development effort as indicated in the methodology section. Generally, the metric of commit quantity is no quantitative measure of the work effort, but gives an abstract approximation through the frequencies of occurrences. If authorization was touched in a development step, a developer needed to invest cognitive effort. To rule out issues from large and infrequent in contrast to small and frequent commits, we also analyzed the number of enforcement changes per commit. While means on the change quantities are problematic because of our non-exclusive coding, the categories of enforcement changes showed similar patterns of commits with high (up to 40 changes, six commits with more than 20 changes) and small numbers of changes. The large ones were mostly merge commits, encompassing several changes in one commit. On the other hand, the combination of changes into single commits occurred at random and should thus not have impacted the validity of the ratios that were the foremost source for the findings.

## 5 Advice on Authorization Enforcement Design

From the findings in the case study and the analysis of the factors that lead to the results, we derive advice on the design of authorization enforcement. First, an API should facilitate the use of universally applicable enforcement statements that provide enough context for unanticipated authorization requirements, such as the accessed “branch” object to decide whether the activity “read this branch” is allowed. As demonstrated in the study, if the context is not meaningful enough, enforcement changes may be necessary when authorization requirements change.

Second, it helps when the same authorization rules can be used in multiple places. One example relates to associated objects and checking whether accessing a list of objects with a specific limitation is permitted

as in “list branches of this company”, re-using the specific permissions for “read branch”. Another example is the defense-in-depth design principle of placing enforcement hooks on several levels. Defense-in-depth only requires limited additional implementation effort when the authorization decision is derived from one policy specification for multiple layers. The study shows that policy changes can thus be reduced.

Third, it is important for the enforcement statements to only pose a low implementation threshold and, thus, encourage implementing enforcement early-on. One approach is the “convention over configuration” paradigm, for example matching object types to controller names or permission names to the CRUD activities. The result is less code, improved readability and reduced redundancy, but a slightly steeper learning curve.

## 6 Conclusion

In this paper, we show that authorization can have a significant impact on software development and that loose coupling of authorization enforcement and policy specification improves authorization development. Based on the case study, we formulated advice on the enforcement API design. We indicated several limitations of the case study, mostly concerned with the project selection and the methodology. Still, the results should be relevant to a broader audience, given that previous publications are thin on authorization aspects in software development. Moreover, we are confident that the key findings, while in need of further validation in additional studies, already provide added value for authorization development. As part of this work, we also describe a methodology for authorization development analysis, facilitating the validation of the results in further development context and with other authorization frameworks in future work. Part of these efforts will be to study the differences to commercial off-the-shelf software development where policy specification is only added after release at deployment time.

## References

1. Gail-Joon Ahn, Longhua Zhang, Dongwan Shin, and B. Chu. Authorization management for role-based collaboration. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 5, pages 4128–4134, Oct. 2003.
2. James P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Deputy for Command and Management Systems, L.G. Hanscom Field, Bedford, MA, October 1972.

3. Steffen Bartsch. Supporting authorization policy modification in agile development of Web applications. In *Fourth International Workshop on Secure Software Engineering (SecSE 2010)*. IEEE Computer Society, 2010.
4. Steffen Bartsch, Karsten Sohr, and Carsten Bormann. Supporting Agile Development of Authorization Rules for SME Applications. In *3rd International Workshop on Trusted Collaboration (TrustCol-2008)*. Springer, 2009.
5. Elisa Bertino, Elena Ferrari, and Vijay Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, 2(1):65–104, 1999.
6. Konstantin Beznosov, Yi Deng, Bob Blakley, and John Barkley. A resource access decision service for corba-based distributed systems. *Computer Security Applications Conference, Annual*, 0:310, 1999.
7. Sacha Brostoff, Martina Angela Sasse, David W. Chadwick, James Cunningham, Uche M. Mbanaso, and Sassa Otenko. 'R-What?' development of a role-based access control policy-writing tool for e-scientists. *Softw., Pract. Exper.*, 35(9):835–856, 2005.
8. Paul Cairns and Anna L. Cox. *Research methods for human-computer interaction*. Cambridge Univ. Press, Cambridge, 2008.
9. Steven Clarke. Measuring API usability. *Dr. Dobb's Journal*, May 2004.
10. Charles Consel and Renaud Marlet. Architecture software using: A methodology for language development. In Catuscia Palamidessi, Hugh Glaser, and Karl Meinke, editors, *PLILP/ALP*, volume 1490 of *Lecture Notes in Computer Science*, pages 170–194. Springer, 1998.
11. Bart De Win, Frank Piessens, Wouter Joosen, and Tine Verhanneman. On the importance of the separation-of-concerns principle in secure software engineering. In *ACSA Workshop on the Application of Engineering Principles to System Security Design*, 2003.
12. David Ferraiolo and Richard Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.
13. Joseph A. Goguen and José Meseguer. Security policies and security models. *Security and Privacy, IEEE Symposium on*, 0:11, 1982.
14. Li Gong and Gary Ellison. *Inside Java(TM) 2 Platform Security: Architecture, API Design, and Implementation*. Pearson Education, 2003.
15. Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, 1976.
16. Almut Herzog and Nahid Shahmehri. A usability study of security policy management. In *Security and Privacy in Dynamic Environments (SEC)*, volume 201, pages 296–306. Springer, 2006.
17. Philip Inglesant, M. Angela Sasse, David Chadwick, and Lei Lei Shi. Expressions of expertness: the virtuous circle of natural language for access control policy specification. In *SOUPS '08: Proceedings of the 4th symposium on Usable privacy and security*, pages 77–88, New York, NY, USA, 2008. ACM.
18. Trent Jaeger, Antony Edwards, and Xiaolan Zhang. Consistency analysis of authorization hook placement in the linux security modules framework. *ACM Trans. Inf. Syst. Secur.*, 7(2):175–205, 2004.
19. Maritza Johnson, Steven Bellovin, Robert Reeder, and Stuart Schechter. Laissez-faire file sharing. In *New Security Paradigms Workshop 2009*, 2009.
20. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP'97 Object-Oriented Programming*, pages 220–242, 1997.

21. Andrew J. Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 344–353, Washington, DC, USA, 2007. IEEE Computer Society.
22. Meir M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.
23. Raju Pandey and Brant Hashii. Providing fine-grained access control for java programs. In Rachid Guerraoui, editor, *Proceedings of 13th European Conference Object-Oriented Programming (ECOOP'99)*, volume 1628 of *Lecture Notes in Computer Science*, pages 449–473. Springer, 1999.
24. John F. Pane, Chotirat Ann Ratanamahatana, and Brad A. Myers. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, 54(2):237–264, 2001.
25. Robert W. Reeder, Clare-Marie Karat, John Karat, and Carolyn Brodie. Usability challenges in security and privacy policy-authoring interfaces. In Maria Cecilia Calani Baranauskas, Philippe A. Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa, editors, *INTERACT (2)*, volume 4663 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 2007.
26. Jackie Rees, Subhajyoti Bandyopadhyay, and Eugene H. Spafford. Pfires: a policy framework for information security. *Commun. ACM*, 46(7):101–106, 2003.
27. Pierangela Samarati and Sabrina de Vimercati di Vimercati. Access control: Policies, models, and mechanisms. *Foundations of Security Analysis and Design*, pages 137–196, 2001.
28. Karsten Sohr and Bernhard Berger. Idea: Towards architecture-centric security analysis of software. In Fabio Massacci, Dan S. Wallach, and Nicola Zannone, editors, *ESSoS 2010*, volume 5965 of *Lecture Notes in Computer Science*, pages 70–78. Springer, 2010.
29. Bernard Stepien, Stan Matwin, and Amy Felty. Strategies for reducing risks of inconsistencies in access control policies. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES2010)*. IEEE Computer Society, 2010.
30. Jeffrey Stylos, Steven Clarke, and Brad Myers. Comparing API design choices with usability studies: A case study and future directions. In *Proceedings of the 18th Workshop of the Psychology of Programming Interest Group*, 2006.
31. Anneliese von Mayrhauser and A. Marie Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, aug. 1995.
32. Alma Whitten. *Making Security Usable*. PhD thesis, CMU, 2004. CMU-CS-04-135.
33. Xinwen Zhang, Sejong Oh, and Ravi Sandhu. PBDM: a flexible delegation model in RBAC. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 149–157, New York, NY, USA, 2003. ACM.
34. Mary Ellen Zurko, Rich Simon, and Tom Sanfilippo. A user-centered, modular authorization service built on an RBAC foundation. In *IEEE Symposium on Security and Privacy*, Los Alamitos, CA, USA, 1999. IEEE Computer Society.