Templates for Misuse Case Description

Guttorm Sindre[1], Andreas L. Opdahl[2]

[1] Dept of Computer and Information Science, Norwegian Univ. of Science and Technology
guttorm@idi.ntnu.no
[2] Dept of Information Science, Univ. of Bergen, Norway
andreas@ifi.uib.no

Abstract. Use cases have proven helpful for eliciting, communicating and documenting requirements. But whereas functional requirements are well supported, use cases provide less support for working with extra-functional requirements, such as security requirements. With the advent of e-commerce applications, security and other extra-functional requirements are growing in importance. In an earlier paper, the authors have introduced the concept of misuse cases – inverted use cases to denote functions that should not be possible to perform in a system. In this paper, security related misuse cases are elaborated in further detail through a discussion of templates for their textual description. **Keywords: Use cases, scenarios, requirements, extra-functional requirements, security.**

1 Introduction

Use cases [1], [2], [3] have proven helpful for the elicitation of, communication about and documentation of requirements [4], [5], [6], [7]. Many stakeholders feel more comfortable with descriptions of activity paths than less operational SRS's focusing on "The system shall..." requirements. The explicit focus on "actors" (which are really roles) further aids communication with end-users, and simple and intuitive diagrams may provide nice overviews of system functionality.

But there are also problems with use case based approaches to requirements engineering [2], [8], [9], [10]. Important requirements may be missed because of oversimplified assumptions about the problem domain, and there are often premature design decisions, especially concerning the user interface. Partly, this may be due to problems with the process followed in the particular projects studied. But there are also intrinsic problems with use cases, in particular they are not equally suited for all kinds of requirements.

A use case typically describes the interaction between user and system to achieve some wanted function. Thus, use cases are good for working with so-called functional requirements, but not necessarily with extra-functional ones, such as security requirements [11], [12]. These will seldom be stated directly by the stakeholders, who rather have concerns about what should not happen in the system [13], [14]. Use cases, by their nature, concentrate on what the system should do, and have less to offer when describing the opposite. But system behavior that should be avoided is also behavior, which could potentially be investigated through use cases. This motivated the extension of use case concepts proposed in [11]:

- A misuse case is the inverse of a use case, a function that the system should not allow. In more detail it might be defined as a completed sequence of actions which results in loss for the organization or some specific stakeholder.
- A mis-actor is the inverse of an "actor", someone who – intentionally or accidentally – initiates misuse cases and whom the system should not support in doing so.

Apart from suggesting these concepts, [11] mainly looked at notation, as shown in Figure 1. This figure depicts part of the functionality for an e-commerce system. Misuse cases are shown as inverted use cases, and mis-actors similarly as inverted actors.

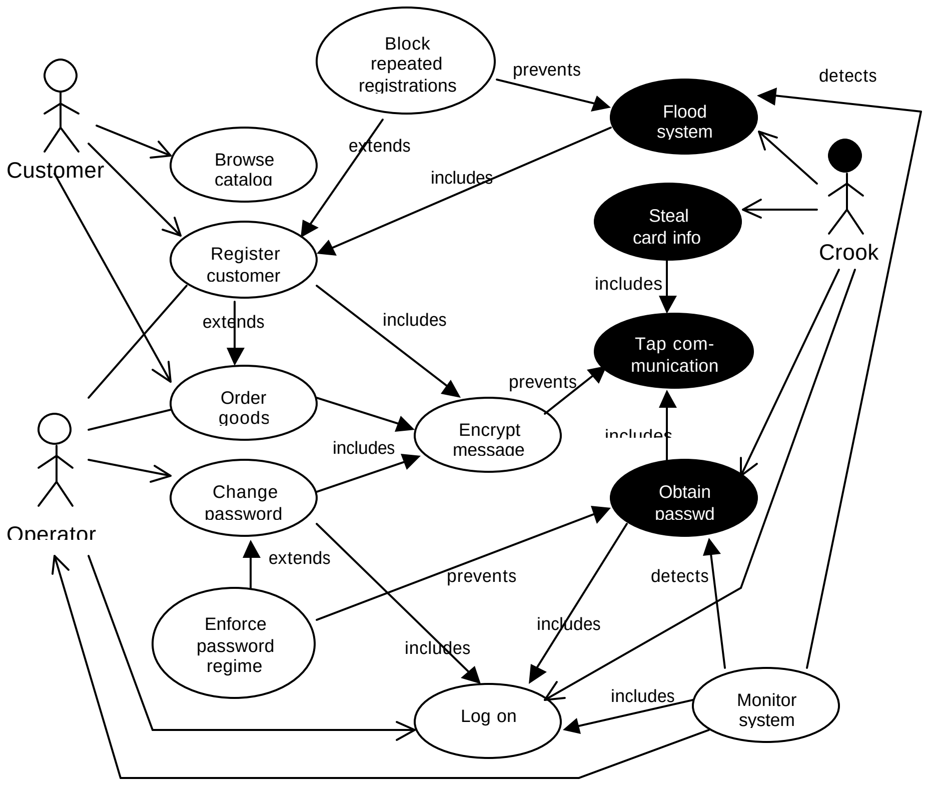

Figure 1: Misuse depicted as “inverted” use cases.

There may be various relations between ordinary use cases and misuse cases. First, there can be an "includes" or "extends" relation from a misuse-case to an ordinary use case, showing that some function for ordinary use is utilized for misuse. This will often be the case. For instance, a denial of service (DoS) attack need not include illegal actions, just flooding the system with a heavy burden of publicly available registration requests. Two other relations were introduced in [11], "prevents" and "detects", which go from ordinary use cases to misuse cases, to indicate functions that prevent or detect misuse. Because of the important relations between use and misuse, it is useful to depict them in the same diagram. Then, a clear difference in notation is needed to avoid confusion, as provided by the given proposal.

However, as stated in [3], use case *diagrams* only give an overview -- the essence of use case modeling is in the textual representation. The topic of this paper is to move on to a discussion of the textual representation of misuse cases. There are many topics that need to be dealt with in that respect -- style, content, method guidelines etc. In this paper we restrict ourselves to discussing *templates* for the textual representation of misuse cases. Some might say that a discussion of method guidelines for writing good activity paths would be more important than looking at templates covering all kinds of extra information that goes into the use case. Our motivation for starting out with templates, though, is as follows:

- If any kind of experiment is going to be done (for instance with students writing misuse cases), most of these will prefer to have clear information about the format of the documentation produced.
- A template may in itself help to write clear and simple paths. If other kinds of information (assumptions, preconditions, …) can be placed in separate fields, the writers are less likely to clutter the paths themselves with such complicating information.

The rest of the paper is structured as follows: Section 2 reviews some templates for normal use cases. Section 3 then discusses adaptations of these templates to capture misuse cases. Section 4 outlines a method for using the new templates. Section 5 discusses the feasibility of the approach and its relation to other work. Section 6 concludes the paper.

2 Templates for use cases

Various templates have been suggested for the textual description of use cases. Kulak and Guiney [5] suggests a template consisting of the following parts:

- Use Case Name: A simple, intuitive name that uniquely identifies the use case.
- Iteration: *Facade, filled, focused, finished* - denoting how refined the description is.
- Summary: One or two sentences describing the interaction.
- Basic course of events: The steps that the actors and the system go through to accomplish the goal of this use case. This is "the most common path taken", where no errors occur and the goal of the use case is reached.
- Alternative paths: Less common paths than the basic course.

- Exception paths: Paths taken when errors occur.
- Extension points: Steps in the use case from where extending use cases diverge.
- Triggers: The entry criteria for the use case, i.e., what initiates it.
- Assumptions: Conditions assumed *true* for normal execution of the use case. But contrary to preconditions, assumptions cannot be guaranteed by the system itself.
- Preconditions: Conditions that must be true before the use case can be performed.
- Postconditions: What will be true when the use case is completed. According to [5], this should cover any alternative path, but not necessarily every exception path.
- Related business rules: Use cases describe the system boundary in an operational manner. This can be coupled to more declarative business rules.
- Author and Date.

The template suggested by Cockburn [3] has some interesting deviations from the above, although the most evident fields are overlapping. Most notably, some extra fields are introduced:

- Primary actor: The user who initiates or performs the major actions in the use case.
- Scope: The scope of modeling, e.g., the entire business or just the planned computerized information system.
- Level: What level of abstraction this use case is on. This can be, e.g., summary, user goal, or sub-function.
- Stakeholders and interests: Listing the various stakeholders and what their motivations are.
- Technology and Data variations list: These are for various ways to do the same thing (e.g., successfully paying by cash, check or credit card). Kulak and Guiney use alternate paths for this. In Cockburn's view, there is an advantage in not having to write full paths for all such variations. Besides, one does not have to selecting e.g. one payment method as the "normal" one if really all three are ok.

The template suggested by the Rational Unified Process [15] contains many of the same entries. Its basic form runs 1. Use Case Name, 1.1 Brief Description, 1.2 Actors, 1.3 Triggers; 2. Flow of events, 2.1 Basic Flow, 2.2 Alternative Flows, 2.2.1 Condition 1, (what to do), 2.2.2 Condition 2 ..., etc.; 3. Special Requirements, 3.1 Platform ..., ...; 4. Preconditions, 5. Postconditions, 6. Extension Points. As can be seen, this is quite similar to Kulak and Guiney's template. The most notable deviation is the inclusion of a section for special requirements, such as platform requirements. The RUP template has been criticized (e.g., by [3]) for cluttering the use case description with excessive numbering of the various sections.

There are several other templates for use cases, but for the purpose of misuse cases it is not necessary to look at all these – the goal here is not to decide what is the best use case template, but to get a first overview of what template fields will be useful for misuse cases.

3 Adaptations for misuse cases

Some items that are common to all the three above-mentioned templates are obviously relevant for misuse cases, such as Name, Summary, Author, Date, and Basic Path (the normal path of action taken, ending with success for the mis-user / failure for the system). These will be included in our template without further discussion. For other items that also seem relevant, there are discrepancies between the templates we looked at in the previous section. Hence, some discussion is mandated:

Extensions/alternative paths/exceptional paths: Here Kulak and Guiney have three fields (alternative paths, exception paths, and extension points.) According to their definition, exception paths are paths where some error occurs, whereas alternative paths are paths where everything proceeds normally. Cockburn partly covers the alternate paths with his Technology and Data Variations List, but in a more abstract manner. Cockburn's approach has an advantage in avoiding premature design considerations. But for misuse cases it may be of particular interest exactly to highlight specific technologies or extreme data values that can be exploited. Exceptions/extensions are also meaningful for use cases, representing the various ways in which misuse is prevented or detected, cf. fig. 1. All in all, we suggest the following fields here:

Name: Obtain Password
Summary: A crook obtains and later misuses operator passwords for the e-shop by tapping messages sent through a compromised network host during operator log on.
Author: David Jones
Date: 2001.02.23.

Basic path:
bp0 A crook has hacked a network host computer and installed an IP packet sniffer (step bp0-1.) All sequences of messages sent through the compromised host and which contain strings like 'Logon', 'User name', 'Password', 'passwd' etc. are intercepted and analysed further (step bp0-2 and extension point e1.) In this way, the crook collects (likely) user names and passwords along with the IP addresses of the computers they are valid on (step bp0-3.) The crook - possibly much later - uses the user name and password to gain illegal operator access to the e-shop computer (step bp0-4.)

Alternative paths:
ap1 The crook has operator privileges on the network host. No hacking of the network computer is necessary (changes step bp0-1.)
ap2 The crook has not penetrated a network host, but instead intercepts messages sent through the telephone system from the e-shop operator's home (changes step bp0-1.)
ap3 Instead of home phone, the crook intercepts messages sent from the e-shop operator's portable devices (changes step bp0-1.)

Capture points:
cp1 The password does not work because it has been changed (in step bp0-4.)
cp2 The password does not work because it is time dependent (in step bp0-4.)
cp3 The password does not work because it is different for different IP addresses (in step bp0-4.)
cp4 Operator logon to the e-shop is only possible from certain IP addresses (in step bp0-4.)
cp5 Communication tapping (in step bp0-2) is not possible (perhaps because the communication is encrypted.)

Extension points:
ep1 Includes misuse case "Tap communication" (in step bp0-2.)

Table 1: Detailed misuse case description, part 1.

- Alternative paths: Various ways the misuse can be done.
- Capture points: Options for how the misuse may be prevented and/or deteced at particular steps.
- Extension points: Optional paths may be included in the misuse case in some situations. These will be options taken by the mis-user, for instance to hack around hindrances, whereas capture points work against the mis-user.

Several options can be mentioned in the same field, for instance known or suggested options and routines for dealing with the misuse. A detailed description of how this is done will often be more of a design task, though.

Conditions: Here, Kulak and Guiney have three somewhat related fields: Triggers (entry criteria, what initiates the use case), Assumptions (conditions which must be

Triggers: tr1 Always true, i.e., this can happen at any time.

Preconditions:
pc1 The system has a special user 'operator' with extended authorities.
pc2 The system allows the operator to log on over the network.

Assumptions:
as1 The operator uses the network to log on to the system as operator (for all paths.)
as2 The operator uses his home phone line to log on to the system as operator (for ap2.)
as3 The operator uses his home phone line to log on to the system as operator (for ap3.)

Worst case threat (postcondition):
wc1 The crook has operator authorities on the e-shop system for an unlimited time, i.e., she is never caught.

Capture guarantee (postcondition):
cg1 The crook never gets operator authorities on the e-shop system.

Related business rules:
br1 The role of e-shop system operator shall give full privileges on the e-shop system, the e-shop system computer and the associated local network host computers.
br2 Only the role of e-shop system operator shall give the privileges mentioned in br1.

Potential misuser profile: Highly skilled, potentially host administrator with criminal intent.

Stakeholders and threats:
sh1 e-shop
- reduced turnover if misuser uses operator access to sabotage system
- lost confidence if security problems get publicized (which may also be the misuser's intent)

sh2 customer
- loss of privacy if misuser uses operator access to find out about customer's shopping habits
- potential economic loss if misuser uses operator access to find credit card numbers

Scope: Entire business and business environment.
Abstraction level: Mis-user goal. **Precision level:** Focussed.

Table 2: Detailed misuse case description, part 2.

true but which cannot be guaranteed by the system itself) and Preconditions (which can be ensured by the system itself.) The trigger field is needed to take care of situations where something else than the primary actor initiates the use case. For misuse this may be interesting, e.g., in connection with viruses triggered by timing. The fields Assumptions and Preconditions are useful for misuse cases in much the same manner as for ordinary use cases. When it comes to postconditions, Kulak and Guiney has this one field, whereas Cockburn has more, distinguishing between minimal guarantees and success guarantees. For misuse cases, it may be useful to distinguish between guarantees for the misuser and guarantees for the system/stakeholders. All in all we end up with the following fields relating to conditions:

- Trigger: The condition that initiates the misuse case. In many cases, this may just be the predicate True, indicating that some danger is permanently present.
- Assumptions: This condition describes those states of the system's environment that make the misuse case possible.
- Preconditions: This condition describes those states of the system that make the misuse case possible.
- Worst case threat: Describes the outcome if the misuse succeeds. If the misuse case has alternative paths, this condition will often be or contain a disjunction to describe slight variations in the outcome.
- Prevention guarantee: Describes the guaranteed outcome whatever prevention path is followed. If no prevention path is followed, one might alternatively formulate a wanted prevention guarantee, expressing what one would want the system to achieve with respect the attempted misuse, but without stating how.
- Detection guarantee: Describes the guaranteed outcome whatever detection path is followed. As above, one might also make this a wanted detection guarantee.
- Related business rules: This is interesting for misuse cases, just as for use cases. It is useful to see exactly what business rules are broken by each misuse case, and possibly discover situations where the business rules themselves are too weakly formulated, thus opening for misuse.

These various fields may be useful for further security analysis and prioritization of use cases. For instance, assume that the project team has decided that the system should support some easy-to-implement and nice-to-have (but not strictly necessary) use case UC1. But what if UC1 has overlapping assumptions and/or preconditions with a really dangerous misuse case MUC1 — and is the only use case with such overlap. If so, MUC1 could be prevented quite easily if UC1 was not implemented. Then, the full cost of UC1 is a lot bigger than the mere resources for its implementation, as one must also add the potential cost of MUC1 happening, or perhaps of some expensive countermeasures to MUC1.

Other fields: The remaining fields of our suggested template can be discussed somewhat more briefly:

- Iteration: Obviously, there is a need also for both superficial and more detailed descriptions. The best thing to do is probably to use the same levels as you do for normal use cases.
- Misuser profile: Cockburn has the field Primary actor, normally associated with a certain role in the organization. For a misuse-case the primary mis-actor may be a totally unknown. Nevertheless, the field can be useful for stating whatever there is to state about the mis-actor. For instance, some kinds of misuse are most likely to be performed by intent whereas other may happen accidentally. Some require insiders or people with high technical skill, others not.
- Scope: This field represents the scope of modeling, e.g., an entire business, an information system of users and computers, or just the computerized information

system. All these scopes are important. The defenses against misuse attempts may be of a physical or organizational nature, in addition to designing security into the computerized information system. The computerized system is no more secure than its physical and organizational environment.

- Level: This field indicates what level of abstraction a use case is on. This can be e.g., summary, user goal, or sub-function. Misuse cases can be specified at several levels of abstraction just like normal use cases.
- Stakeholders and Risks: For normal use cases, Cockburn calls this field Stakeholders and Interests. On an abstract level, risks could simply be described textually. With more ambition one might try to quantify misuse likelihoods and costs. Here, conventional risk and hazard analysis techniques would come into play, but that is beyond the scope of this paper. The interplay between such more formal analyses and misuse cases is thus left to further work.
- Technology and data variations: This list makes it possible to mention some variations without giving a path for each of them. E.g., a misuser may access the system either through a PC web browser or a WAP phone, but apart from detailed actions related to the particular equipment, the misuse steps in these two cases will be the same. In this case it may feel unnecessary to list two separate paths for this, instead the various equipment may simply be listed.

4 Method issues

We have suggested a template with quite a number of fields, which will often seem over-elaborate early on, when one is just trying to get an overview of various use- and misuse cases. However, the template is just a suggestion for fields that a misuse case description should contain in its finished state. Along the way, most of the fields would not be mandatory. Just like a use case, a misuse case could be presented with only a name and brief description, along the lines of the "casual" version suggested in [3] or the "facade" iteration suggested in [5], allowing for what is traditionally called specification freedom [17]. Something more about sequencing:

1. First concentrate on the normal actors and the main use cases requested by these, i.e., the services that the users want. The motivation for this is quite simple: If there is nothing to use, there is nothing to misuse — hence it would be difficult to start with misuse cases.
2. Then introduce the major misuse cases, i.e., threats that are likely, and mis-actors who might be behind these threats.
3. Investigate the potential relations between misuse cases and use cases, in terms of potential “includes”-relations. This step is quite important since many threats to a system can be achieved through normal system functionality, as for instance with DoS attacks.

We do not generally recommend early investigation of prevents or detects relations, especially not step descriptions of these, which would easily lead to premature design decisions. This is actually the nice thing about misuse cases — it makes possible, so to speak, requirements capture in "negative space". With only positive use cases at hand, the modeler would have to model countermeasures directly. With misuse cases one can simply state that "this is something that must be avoided" and then leave it to more detailed analysis and design to decide how. A valuable input to this decision would be various known or suggested defenses listed in the “capture points” field of our template, which could then be analyzed more thoroughly for coverage and cost [18].

Most of our discussion here suggests that misuse case analysis will be used early on in the development process. But it might also be an idea to redo misuse case analysis on a more detailed level *after* the system’s security defenses have been chosen (or preferably, tentatively chosen) – this may provide an opportunity to paper test the choice of defenses and try to find weaknesses. As stated in [12], when some defenses are introduced, potential attackers will look for other openings – and may find some where it was quite unsuspected. Applying misuse case analysis on several levels of abstraction and at several stages during the development process may increase the chance of eliminating such unpleasant surprises.

5 Discussion

The proposed template for detailing misuse cases supports representation of information relevant for security considerations during requirements determination. The proposed approach could also be helpful in the early elicitation of security requirements in several other ways:

- Early focus on security. Security issues are often considered a design-level — and sometimes even a programming-level — issue. This is unfortunate, because the requirements stage is where the appropriate security issues can be identified, analyzed and balanced against one another.
- User/customer assurance. End-users and management of the procuring organization may not be able to discuss the technical details of security threats and countermeasures. But they will have security *concerns*. Seeing these captured by misuse cases will reassure them that the problem is actually being dealt with.
- User/customer awareness. Security is as much about organizational routines as technical defenses, and even the best defense will fail if the awareness about security threats is low. Misuse cases can help to keep more of the discussion about security in a format that end-users may understand, and thus learn from.
- Analyst creativity. Security analysts need to be as creative as potential computer criminals (or as authorized users making unforeseen mistakes!) to identify the relevant threats beforehand. We believe that the explicit writing of misuse cases can stimulate such creativity.
- Traceability. If countermeasures are described as first-class requirements, one easily misses out on the motivation for these countermeasures. The explicit registering of misuse cases can show *why* a certain defense was introduced in the system, which will help later maintenance and redesign in the face of changing threats.

- Requirements organization. A major problem in managing extra-functional requirements is how to organize the requirements document and relate functional and extra-functional requirements to one another. Misuse cases solve this problem for a large class of extra-functional requirements. As mentioned previously, one may detect that a use case has the same assumptions and preconditions as a misuse case. This is a clear hint to investigate whether that use case is strictly necessary. Such analysis might resemble the use of root requirements in [18].
- Transition to object-oriented design. An additional advantage is that the application of use cases also for security-intensive systems, makes it possible to build on all the ongoing work providing transitions between this and object-oriented design
- Uniform handling of functional and extra-functional requirements. As mentioned in the introduction, misuse cases are applicable and useful for dealing with a larger class of extra-functional requirements. Security requirements are only one example. Extra-functional requirements dealing with *safety*, *availability* and *robustness* all state what should *not happen* and therefore fall into this class. As a consequence, misuse cases offer a uniform way of handling them. This is significant, because a major obstacle to dealing with extra-functional requirements is their inherent diversity, which calls for a broad array of techniques to be used, but which makes the resulting requirements document difficult to organize and comprehend.

There are several methods that somehow utilize use cases or scenarios in requirements capture, for instance [20], [21], [22], [23]. On the other hand, methods for security requirements engineering have traditionally not been use case based, e.g., [24]. In [25] there is a discussion of using scenarios for developing secure e-commerce systems, but not suggesting any concept resembling the misuse case. Hence, this is believed to be a novel concept, and our investigations indicate that it can be a quite useful extension to the repertoire of the requirements engineer.

But misuse case analysis alone makes no requirements engineering technique. Methods such as the ones referred to above have many years of work behind them and are a lot more detailed than our misuse case analysis with respect to method guidelines and tool support. We only look at misuse analysis where other methods address scenarios/use cases and goal structures in general. Moreover, this paper has only looked at *templates* for misuse cases, while the quality of what is entered in the various fields may be more important, in particular what is entered in the basic and alternative paths. The issue of how to write good misuse case paths has not at all been discussed here. Hence, what we have proposed so far must be elaborated with further method guidelines for writing good activity paths, e.g., inspired by [2], [3], [26], [27] for normal use cases, as well as integrated with more full-fledged RE methods such as the ones mentioned in the previous paragraph.

6 Conclusions and further work

Use case diagrams are often good for eliciting functional requirements, but not so good for security requirements, which relate to activities *not* wanted in the system. We have provided a diagram notation and a textual template for misuse cases. The approach has been tested on several small examples and is used in ongoing research investigating the use of *security patterns* in requirements work. However, misuse cases must now be evaluated in an industrial setting. We think the proposed approach is well suited for such evaluations:

- The approach is close to standard UML use case diagrams, which are already heavily used in industry.
- Security requirements are increasingly important with the advent of e-commerce, and problems with use cases here *will* thus be an industrial problem.

As the discussion has revealed, many template fields used for normal use cases are relevant for misuse cases, too, and probably, much methodology for use cases can be adapted to misuse cases. Interesting further work will be to integrate misuse case analysis with other use case based and goal based RE approaches. Other interesting directions to pursue are integration with more traditional techniques for risk analysis and costing, which have been popular in security and safety engineering. Safety might be an interesting direction to widen the application of misuse cases – again to describe things that should not happen in a system. As stated by [28] a better integration between informal and formal techniques is needed to make progress in safety analysis. Anyway, it should be noted that misuse cases is not intended as a standalone technique to solve security (or safety) problems by itself – it must be integrated with other RE techniques and a more formal analysis. However, we believe that misuse cases have advantages when it comes to communication with end-users, and can thus enhance other techniques.

.

References

1. I. Jacobson et al., Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.
2. L. L. Constantine and L. A. D. Lockwood, Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design, ACM Press, 1999.
3. A. Cockburn, Writing effective use cases, Addison-Wesley, 2001.
4. J. Rumbaugh, “Getting Started: Using use cases to capture requirements”, Journal of Object-Oriented Programming, September 1994, pp. 8-23.
5. D. Kulak and E. Guiney, Use Cases: Requirements in Context, ACM Press, 2000.
6. M. Arnold et al., "Survey on the Scenario Use in Twelve Selected Industrial Projects", technical report, RWTH Aachen, Informatik Berichte, Nr. 98-17, 1998.
7. K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer, "Scenario Usage in System Development: A Report on Current Practice", IEEE Software, 15(2): 34-45, March/April 1998.
8. A. I. Antón, J. H. Dempster, and D. F. Siege, “Deriving Goals from a Use Case Based Requirements Specification for an Electronic Commerce System”, Proc. REFSQ’2000.
9. J. Arlow, “Use Cases, UML Visual Modelling and the Trivialisation of Business Requirements”, Requirements Engineering Journal, 3(2):150-152, 1998.

10. S. Lilly, “Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases”, Proc. TOOLS-USA’99, pp.174-183, 1-5 Aug 1999.
11. G. Sindre and A. L. Opdahl, "Eliciting Secutiry Requirements by *Mis*use Cases", Proc. TOOLS Pacific 2000, pp 120-131, 20-23 Nov 2000.
12. C. P. Pfleeger, Security in Computing, Prentice-Hall, 1997.
13. G. Kotonya and I. Sommerville, Requirements engineering: Processes and Techniques, Wiley, 1997.
14. P. Loucopoulos and V. Karakostas, Systems Requirements Engineering, McGraw-Hill, 1995.
15. P. Kruchten: The Rational Unified Process – an Introduction, Addison-Wesley, 2000.
16. R. Wirfs-Brock, "Designing Scenarios: Making the Case for a Use Case Framework", Small-talk Report, November-December 1993.
17. P.E. London and M.S. Feather, “Implementing specification freedoms”, Readings in Artificial Intelligence and Software Engineering, pp. 285-305, Morgan Kaufmann, 1986.
18. C. Irvine, T. Levin, “Toward a Taxonomy and Costing Method for Security Services”, Proc. ACSAC’99, Phoenix AZ, 6-10 Dec 1999.
19. W. N. Robinson, S. Pawlowski, “Surfacing Root Requirements Interactions from Inquiry Cycle Requirements Documents”, Proc. ICRE’98, Colorado Springs, 6-10 Apr, 1998.
20. A. I. Antón, “Goal-Based Requirements Analysis”, Proc. ICRE’96, pp. 136-144, 1996.
21. N. Maiden, S. Minocha, K. Manning, and M. Ryan, “CREWS-SAVRE: Systematic Scenario Generation and Use”, Proc. ICRE’98, pp.148-155, 1998.
22. C. Potts, “A ScenIC: “A Strategy for Inquiry-Driven Requirements Determination”, Proc. RE’99.
23. C. Rolland, C. Souveyet, and C. Ben Achour, “Guiding Goal Models Using Scenarios”, IEEE Transactions on Software Engineering, 24(12): 1055-1071, Dec 1998.
24. J. Kirby Jr, M. Archer, C. Heitmeyer, “SCR: A Practical Approach to Building a High Assurance COMSEC System”, Proc. ACSAC’99, Phoenix AZ, 6-10 Dec 1999.
25. A. I. Antón, J. B. Earp, ”Strategies for Developing Policies and Requirements for Secure Electronic Commerce Systems”, Proc. 1st ACM Workshop on Security and Privacy in E-Commerce, Athens, 1-4 Nov 2000.
26. C. Ben Achour, C. Rolland, N. A. M. Maiden, C Souveyet, "Guiding Use Case Authoring: Results from an Empirical Study", Proc. RE'99, Limerick, Ireland, 1999.
27. K. Cox, K. Phalp, M. Shepperd: "Comparing Use Case Writing Guidelines", Proc. REFSQ'2001, Interlaken, Switzerland, 2001.
28. R. R. Lutz, “Software Engineering for Safety: A Roadmap”, in A. Finkelstein (ed.): The Future of Software Engineering, ACM Press, 2000.