Misuse Cases Help to Elicit Non-Functional Requirements

Ian Alexander
Independent Consultant
Ian.Alexander@scenarioplus.org.uk

Abstract

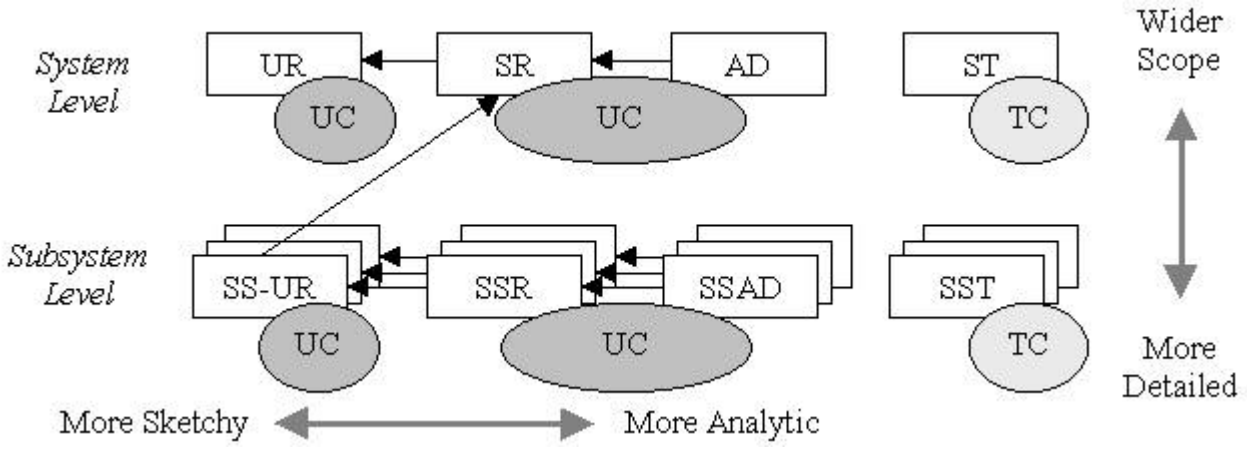
Use Cases are widely seen as suitable for defining functional requirements for software. There is controversy about the suitability of Use Cases for Systems other than Software, and for Non-Functional Requirements (NFRs). This centres on the variety of methods proposed for use case analysis, the range of uses proposed for use cases in different contexts, and the perceived imprecision in some of the proposed approaches.

This paper suggests that apparently diverse approaches [Sindre & Opdahl 2001, Allenby & Kelly 2001] can be combined to facilitate different aspects of Systems Engineering (SE). Suitably documented, Use Cases can help to elicit requirements of different kinds for Real-Time and Safety-Related Systems, including not only functions but Reliability, Safety, and Security NFRs. The proposed approach analyses Use and Misuse Cases in a game-like sequence to discover successively more detailed threats and NFRs to mitigate these threats. Simultaneously, systems analysis discovers functions, often of subsystems, to meet these NFRs.

The Scenario Plus toolkit has been extended to draw Misuse Cases, and both *threatens* and *mitigates* links between Use and Misuse Cases, from a requirements database automatically. The implemented approach could form part of a scenario-based method of engineering the requirements for hardware/software systems.

Background: Use Cases in Systems Engineering

My colleagues at DaimlerChrysler and I are investigating the appropriate forms of **Use Cases** (principally for functional requirements) in the Systems Engineering life cycle, to be reported in a companion paper [Alexander and Zink 2002]. The diagram summarizes a simple approach that can be applied at any level (system, subsystem, ...) in the life-cycle. Test cases are omitted for clarity. The essential idea is to document system **Functions** in a story-like way by writing **Scenarios**, supplemented by other information such as Preconditions, to organize the requirements. These scenarios become more detailed and more analytic as system development proceeds; they also tend to 'open the box' to explore the interaction of subsystems as these become known, in accordance with **Ivar Jacobson's original concept**, but contrary to some software engineers who assert that Use Cases should always be Black-Box and at Whole-System level.



Diverse types of Use Case Models for eliciting and analyzing functional requirements in the Systems Engineering life-cycle [from Alexander & Zink 2002]

1. Introduction

Systems Engineers are starting to look at Use Cases for a range of purposes within the systems engineering life-cycle [Hruschka & Rupp 2001, Alexander 2001].

The SE life-cycle is larger than that of software engineering (software typically forming a subsystem), which is the domain addressed by most current literature on use cases [e.g. Cockburn 2001, Kulak & Guiney 2000]. These authors, even if they recommend iteration to make Use Case Models more accurate and more detailed, appear to consider that a single model is sufficient for a software project. But a large system composed of many 'systems', such as a passenger aircraft or a warship, must be analysed in successively greater detail in subsystem models to cope with the complexity of the functionality [Simon 1996, Stevens 1998].

The greater range and the fact that embedded and real-time systems are often complex and safety-related means that systems engineers are often skeptical [e.g. Jorgensen 2001, Mills 2001]. This skepticism is unfortunately not unjustified as some software engineers have apparently rushed to apply Use Case methods informally, hoping to avoid the burden of documenting traditional requirements. Others have proposed 'light' methods [e.g. Beck 2000] that may be unsuitable for large systems.

However, a disciplined use of scenarios should help with the elicitation, validation, and reuse of systems requirements, as well as for guiding design and generating test cases. Operational scenarios have indeed long been used for some of these purposes. Scenarios are known to be effective in elicitation and design [e.g. Carroll 1995].

Background: Use & Misuse Cases

Ivar Jacobson introduced Use Cases in the context of his work on large telecommunication systems [Jacobson 1992]. He had the idea of describing a system's desired behaviour by telling a story at just one level of detail, from the point of view of a user or interfacing system (an '**Actor**'). Such a story (a '**Scenario**'), when supported by subsidiary scenarios (for Alternatives and Exceptions) and associated information, he called a **Use Case**.

Guttorm Sindre and Andreas Opdahl took the traditional concept of a '**Negative Scenario**', i.e. a situation desired not to occur by a system's users, and applied it in a Use Case context to create the idea of a **Misuse Case** [Sindre & Opdahl 2000]. This is a goal (possibly supported by one or more scenarios, though unlikely to be worked out in full detail) desired by an agent (not necessarily human) hostile to the system.

Business and military **planners and game-players** are familiar with working out their opponents' best moves as identifiable threats, and countering these when the threats are considered to be sufficiently likely to be worth neutralizing. The Misuse Case and its hostile Actor neatly encapsulate the idea of evaluating '**Black's Best Move**'. From this perspective, it seems natural to consider how to represent threats, mitigations and similar relationships between Use and Misuse Cases, in a form suitable for Systems Engineering.

A negative form of Use Cases, Misuse Cases [Sindre & Opdahl 2000, 2001] also look promising for systems requirements elicitation. Negative scenarios have also long been applied, e.g. in military and commercial operations planning, but they may be less familiar in systems engineering.

In principle an approach combining Use and Misuse Cases can be applied at any level from whole systems down through subsystems to individual equipments and components. This approach is inherently stepwise, so it should dovetail well with both the stepwise decomposition of the SE life-cycle, and with participative inquiry cycle approaches [e.g. Potts 1994, Heron 1996, Alexander 1999].

Functional and Non-Functional Requirements

NFR is simply a blanket term for all requirements that are **not explicitly Functional**. While the concept of a system **Function** is quite sharply-defined – a piece of behaviour exhibited by the system – the same cannot be said for NFRs. Many classifications have been attempted.

One widely-used group of **NFR**s consists of **Reliability, Availability, and Maintainability** (**RAM**), sometimes combined with Safety. RAM requirements are typically applied in real-time automotive, railway and aerospace systems where the continued correct functioning of the system is important. Some engineers refer familiarly to NFRs as **'-ilities'**, since many NFRs effectively name desirable qualities such as reliability, verifiability, and so on. The term **'Constraints'** is also widely used, perhaps most effectively to describe definite limitations on how the system can be built, such as the need to interface to existing systems, to provide certain standard interfaces, or simply on the size, shape, paint finish, weight and similar required properties of systems. Clearly, non-technical aspects such as budget and timescale can also constrain development projects.

Given this diversity, the term NFR, while not especially informative, is at least unambiguous.

One interesting feature is that Misuse Cases seem naturally to lead to Non-Functional Requirements (NFRs] such as for safety and security, whereas Use Cases essentially describe system behaviour in Functional terms (i.e. first you do this, then you do that), possibly with some closely-associated NFRs such as performance targets for specific transactions.

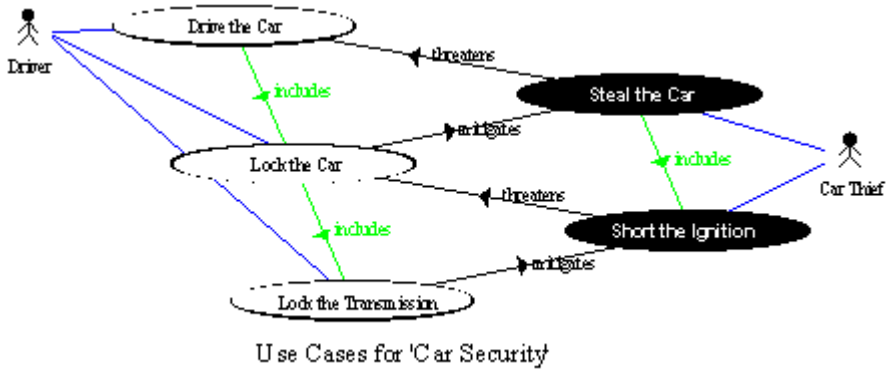
NFRs at high (e.g. whole System) level typically lead to Functions at lower (e.g. Subsystem) levels. For example, a security requirement may lead to security subsystems to protect the system as a whole. Indeed, all security 'systems' can be viewed as subsystems responding to overall system security requirements – nobody wants an alarm unless they have something to protect.

There thus appears to be an important interplay between Use and Misuse cases that could greatly improve the efficiency of eliciting and organizing Functional and Non-Functional Requirements – which themselves are involved in an interplay during system design – in the systems engineering life-cycle. Let us look in turn at how a Use/Misuse Case analysis can derive Security, Safety, Reliability and other Non-Functional Requirements.

2. Use/Misuse Case Analysis – Security Requirements

Security Requirements are caused to exist because people and agents that they create (such as computer viruses) pose real threats to systems in the world. Security is thus unlike all other areas in a specification as someone is consciously and deliberately trying to break the system. The scenarios in which such 'negative' agents attempt to defeat the system under design can be elicited as Misuse Cases in a method proposed by Sindre & Opdahl. The explicitly drawn malign Agents and Misuse Cases (see diagram below) can help to focus attention on the battle against such negative agents and hence can improve system security [Sindre & Opdahl 2000].

Misuse and Use Cases may be developed in stages, going from system to subsystem levels and lower as necessary (*as illustrated by the zigzag pattern in the diagram*). Lower-level cases may highlight aspects not considered at higher levels, possibly forcing re-analysis. The approach is not rigidly top-down but offers rich possibilities for exploring, understanding, and validating the requirements in any direction.



Use/Misuse Case Diagram to elicit Security Requirements. High level is to the left. Use Cases are drawn in white. Misuse Cases are drawn in black.

There is a productive analogy with game-playing here: the 'White' team's best move consists of thinking ahead to the 'Black' team's best move, and acting to block it. For example, if the misuse being threatened is the theft of a car, the White player is the lawful Driver and the Black player is the Car Thief. The driver's freedom to drive the car is at risk if the thief can steal the car. So, the driver needs to be able to lock the car – a derived requirement, which mitigates the threat. This is at the top level of the analysis. The next level is started by considering the thief's response. If this is to break into the car and to short the ignition, thus defeating the lock, a mitigating approach, as for instance locking the transmission or requiring a digital code from the key, is required.

It can be seen informally from the diagram that threat and mitigation form a balanced zigzag pattern of play and counter-play. This 'game' can be reflected in an Inquiry Cycle style of development [Potts 1994, Alexander 1999]. Both Use and Misuse Cases may include subsidiary cases of their own kind, but their relationships to cases of the opposite kind are never simple inclusion. Instead, Misuse Cases threaten Use Cases with failure, and appropriate Use Cases can mitigate known Misuse Cases.

Where such mitigation approaches are already known, development may proceed by selecting which possible design features can be afforded – transmission locks cost money and cannot necessarily be provided on all models of car. So, there is a trade-off between the user requirements (countering misuse) and the design constraints (e.g. cost, weight, size, development schedule). Such trade-offs are possibly more familiar to systems engineers than to software developers.

Where suitable mitigation approaches are not yet known, development and Use/Misuse Case analysis can proceed together, initially but not exclusively top-down. Possible mitigation approaches can be identified, studied, prototyped, evaluated and then selected if suitable. Mitigations may demand new subsystems or components; the existence of these new devices may in turn engender new types of threat. These threats can be analysed in their turn to evaluate the need for further counter-measures. In this situation, analysis and design are intertwined [Swartout & Balzer, 1982] as the design choices crystallize and the system requirements can be stated more specifically.

Security threats are rarely neutralized completely by mitigation measures. Thieves pick locks and break into systems through unsuspected access paths. Partial mitigations are still useful as long as they afford a realistic increase in protection at reasonable cost. The goal of neutralizing all possible threats is of course wishful thinking and cannot be stated as a requirement.

3. Tool Support for Use/Misuse Case Analysis

Considering the automatic handling of Use Cases [Alexander 2001, Scenario Plus 2002] and their interplay with misuse cases, there are thus four situations to consider, namely relationships to and from each kind of case:

		Source Case	
		Use	**Misuse**
Target Case	**Use**	*includes*	*threatens*
	Misuse	*mitigates*	*includes*

Rule governing creation of relationships between Use and Misuse Cases

This table can be interpreted as a four-part rule governing the automatic creation of relationship types according to the sources and targets of relationships between Use and Misuse Cases. The Scenario Plus toolkit has been extended to implement this mechanism to construct links. The requirements engineer names the Use or Misuse Case within a scenario step in either the Primary Scenario or an Alternative Path [Cockburn 2001] and underlines the name. The analysis tool then scans for underlined phrases, and attempts to match them with existing Use or Misuse Cases. For example, the tool fuzzily matches the phrase 'Stealing the Car' with the Misuse Case goal 'Steal the Car' (see screenshot below). If no match is found, the tool asks the user if a new case should be created. The tool then links the including to the included Use or Misuse Case according to the table.

ID	Use Cases	Links to Included Use Cases
UC-30	**2.1.3 Lock the Car**	
UC-31	**2.1.3.1 Primary Scenario**	
UC-35	System automatically Locks the Transmission to prevent the Car thief from Stealing the Car.	UC-49 Lock the Transmission UC-70 Steal the Car

Automatic Creation of links between Misuse and Use Cases,
by searching for underlined use case names with simple fuzzy matching

This mechanism permits engineers to write use case steps simply and readably in natural languages (such as English). The created links are displayed, drawn on the summary diagram as relationships between Use/Misuse Cases (see above), and can be navigated as usual in the requirements database.

Misuse Cases are identified by a simple flag; users can choose to show or hide Misuse Cases (with their actors and relationships) as desired. Misuse Case Actors are automatically drawn on the right of the diagram; the Misuse Cases themselves are drawn (as Sindre & Opdahl suggest) on the right with inverted colours.

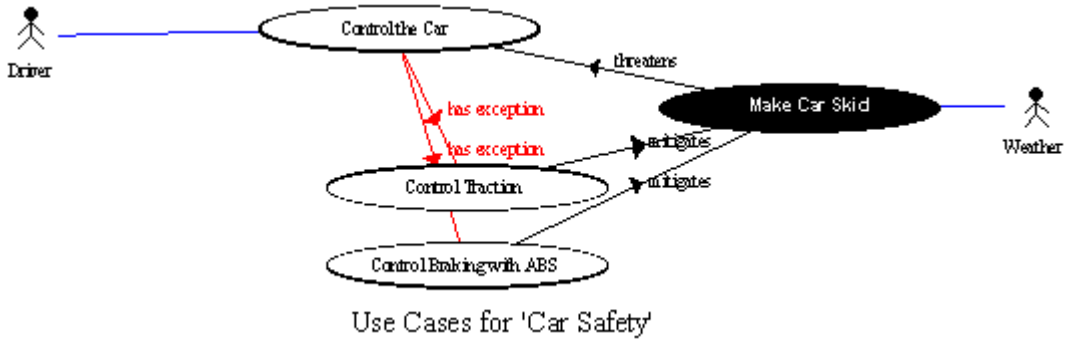
A refinement on Sindre & Opdahl is to permit Misuse Cases to exist at different levels relative to the current context (e.g. the system). Following Cockburn, the levels provided are {Overview, High, Surface, Low, Too Detailed}. For instance, in a system context, a business-wide case would appear as an Overview and might well be considered inappropriate given the setting; the same business case might appear appropriately at surface level in a business model. Accordingly, both Use and Misuse Case levels are shown on scales with high to the left, low to the right. To keep the Misuse Cases apart and on the right of the diagram, Use and Misuse Cases need separate X-axis scales for level, as can be seen on the diagram above.

Mitigation can itself be subclassed into prevention, detection, and remedy [Sindre & Opdahl 2000, 2001]. It is possible to use the names of these subclasses as relationship labels on use case diagrams, but there is a danger of making the diagrams excessively complex [Cockburn 2001]. It may be better therefore to document the precise relationships within the use cases themselves.

4. Safety Requirements from Failure Cases

Allenby & Kelly describe a method for eliciting and analysing safety requirements for aero-engines using what they call 'use cases' [Allenby & Kelly 2001]. They do not suggest the use of negative agents associated with their use cases. Their method is to tabulate the failures, their causes, types, and effects, and then possible mitigations. They observe that mitigations often involve subsystems, i.e. the procedure implies stepwise decomposition. However, since their 'use cases' describe potentially catastrophic failures and their effects, it would seem reasonable to follow Sindre & Opdahl and explicitly call Allenby & Kelly's structures Misuse Cases. 'Failure Cases' is another suitable name.

In the case of safety requirements, there is not generally a human agent posing a threat (though this is possible through sabotage, terrorism, and so on). The agent threatening a negative scenario is typically either the failure of a safety-related device, such as a car's brake, or an inanimate external force such as dangerous weather. For example, a car may become uncontrollable by most drivers if the road is covered in ice or wet leaves. It may be advantageous to anthropomorphise the weather as an agent 'intending' to make the car skid (see diagram below). This uses the force of an easily-understood metaphor [Potts 2001] to emphasize the requirement for control in different weather conditions. Some authors believe that people effectively always operate through metaphor [Byron & Nass 1996, Lakoff & Johnson 1980], in which case it is wise to express requirements in that familiar form.



Use/Misuse Case Summary Diagram for elicitation and analysis of Safety Requirements
'Weather' is anthropomorphized as a malign agent

The use of metaphor and anthropomorphism may appear colourful and even frivolous. However, human reasoning has evolved in a social context to permit sophisticated judgements about possibly hostile intentions of other intelligent agents [Pinker 1997]. Use/Misuse Case analysis deliberately exploits this faculty in the service of systems engineering.

Misuse Cases may help to elicit appropriate solutions in the form of subsystem functions, such as for traction control and automatic braking control. These handle exception events (such as skidding) by carefully programmed responses. These satisfy requirements that may be written as exception-handling scenarios. Such scenarios can either form the Exception subsections of larger use cases (such as 'Control the Car') or may be pulled out into Exception-Handling Use Cases in their own right, as illustrated in the diagram below, where the 'pulling out' is indicated by explicit 'has exception' links.

Once such exception-handling use cases have been identified, the Misuse Cases are not needed except as justification for design decisions. Justifications remain useful to protect design elements and requirements from being struck down at reviews, especially when time and money are scarce. The misuse cases can readily be displayed or hidden as desired by use case tools such as my Scenario Plus toolkit for DOORS [Scenario Plus 2001, DOORS 2001].

As with Security Requirements, Misuse and Use Cases may be developed in stages, going from system to subsystem levels and lower as necessary [Allenby & Kelly 2001]. Again, bottom-up and middle-out working remain possible. The explicit presence of Misuse Cases should make validation of requirements by domain experts easier and more accurate.

5. Other Non-Functional Requirements

It is evident that other NFRs may be handled in a similar way with Use/Misuse Case elicitation and analysis.

Reliability requirements may be elicited and analysed as threats caused by agents of several kinds, including human error, storms, design errors (e.g. software bugs), and metal fatigue. These can cause hardware breakdowns, software crashes and so on. There is a clear relationship between such 'agents' and Exception Classes, which can be used to generate candidate scenarios and hence elicit requirements [Maiden 1998].

Maintainability and Portability requirements may also benefit from the Use/Misuse Case treatment. Here the 'malign agents' could be inflexible design or wired-in device dependence.

It is not difficult to think of example Misuse Cases for other NFRs such as Usability (Simple Simon presses the wrong button), Storability (Jack Frost damages the delicate components), Electromagnetic Compatibility (Jack the Lad's homemade transmitter fries the sensitive electronics) and so on for other '-ilities'.

While I wouldn't advocate blindly creating hundreds of Misuse Cases for every possible requirement, especially when the NFRs in question are well known in advance, the technique does appear to be widely applicable to elicit and justify different types of NFR.

6. Discussion

Far from representing disparate and contradictory approaches to safety, security, and so on, I suggest that systematic application of Use/Misuse Case Analysis in systems engineering can be coherent and beneficial.

Systems engineering can be viewed as the task of saying what you want through scenarios that go into successively more analytic detail ('left to right' from stakeholders to specifications) and that work at successively lower levels ('top to bottom' from systems to components), and designing accordingly [Alexander & Zink 2002]. This process needs to be based on use cases of different types. Such use cases define the functional requirements for each (sub)system addressed, first in black-box and then in white-box detail.

Misuse cases (embodying negative scenarios and malign actors) can enhance this process by identifying and analysing threats to system operation. Hence, by a game-like cycle of play and counter-play, systems engineers can identify appropriate countermeasures as non-functional requirements or constraints. These are likely to trace to subsystems that provide additional functionality to help to meet the non-functional requirement targets.

A system-level summary diagram can show such newly-created countermeasures Use Cases to

indicate the role of subsystems in the design. Those Use Cases will naturally be drawn there at Surface or Low level. In the corresponding subsystem-level analyses, the same use cases will appear at Overview or High level. Such a use case may indeed provide the entire *raison d'être* for a countermeasures subsystem.

The suggested approach is initially but not exclusively top-down. This allows non-functional requirements to be elicited and analyzed in as much detail as is necessary. The interplay between Use and Misuse cases, and between functional and non-functional requirements, promises to be very suitable for participative inquiry cycle approaches. These in turn should lead to greater user involvement and hence better quality requirements and systems.

One possible criticism of Misuse Cases could be that they are little more than headings in a requirements template. Good templates such as Volere [Robertson 1999] are indeed helpful in eliciting and validating requirements simply because they remind us of questions to ask, e.g. 'Could there be any portability requirements here?'. For each template heading or Misuse Case there may be several requirements, but if even one is found – or if it is confirmed that there is no requirement – then the approach is worthwhile. In other words, a template, like a Misuse Case, implies an inquiry method. However, devising threats and malign agents is a more powerful technique for several reasons.

1. By inverting the problem from use to misuse, it opens a new avenue of exploration, helping to find requirements that might have been missed.

2. By asking 'what can go wrong here?', it contributes to searching systematically for exceptions using the structure of the scenarios themselves as a guide [Alexander 2000].

3. By being explicit about threats, it offers immediate justification for the search and indicates the priority of the requirements discovered.

4. By personifying and anthropomorphizing the threats, it adds the force of metaphor, applying the powerful human faculty of reasoning about people's intentions to requirements elicitation.

5. By making elicitation into a game it both makes the search enjoyable and provides an algorithm for the search – white thinks out black's best move, and vice versa. The stopping condition is whether the cost of the mitigation, given its probability of defeating a threat, is justified by the size and probability of occurrence of the threat (cp. Cost/Benefit analysis).

6. By providing a visual representation of threat and mitigation, it makes the reasoning behind the affected requirements immediately comprehensible.

Despite these advantages, some systems engineers may feel that Misuse Cases trivialize serious tasks such as security and safety analysis. But provided the threats identified are credible there is little danger of this. The important element in Use/Misuse Case analysis is thinking out the threats and mitigations for different types of NFR, as has long been done for safety requirements. The summary diagrams are useful in so far as they assist this process of thought, or explain it to other engineers.

Use and Misuse Case engineering offers a promising and solid basis for eliciting and analysing requirements at all levels of hardware/software systems. This is in marked contrast to the prevailing emphasis on Use Cases as tools for object-oriented software development.

References

Alexander, Ian, *Migrating Towards Co-operative Requirements Engineering,* Computing & Control Engineering Journal, Volume 10, Number 1, pp17-22, February 1999

Alexander, Ian, *Engineering Use Cases with DOORS*, Address given at RE'01, IEEE Computer Society p 264, 2001

Alexander, Ian, *Scenario-Driven Search for Exceptions Improves Capture of Process Knowledge,* Proc. DEXA 2000, IEEE Computer Society pp 991-994, 2000

Alexander, Ian and Thomas Zink, *An Introduction to Systems Engineering with Use Cases*, 2002 (to appear)

Allenby, Karen and Tim Kelly, *Deriving Safety Requirements Using Scenarios*, RE'01, pp 228-235, 2001

Beck, Kent, *Extreme Programming Explained*, Addison-Wesley, 2000

Carroll, John M., *Scenario-Based Design, Envisioning Work and Technology in System Development,* John Wiley, 1995

Cockburn, Alistair, *Writing Effective Use Cases*, Addison-Wesley, 2001

DOORS, website (requirements management tool), http://www.telelogic.com 2001

Heron, John, *Co-operative Inquiry: research into the human condition*, Sage, Newbury Park, CA, 1996

Hruschka, Peter and Chris Rupp, *'Echt Zeit' für Use-Cases* (in German), OBJEKTSpektrum, Number 4, pp 63-70, 2001

Jacobson, Ivar, et al: *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992

Jorgensen, Raymond, *The Oxymoron of Use Case Requirements*, INCOSE INSIGHT Newsletter, Volume 4, Issue 2, p 21, July 2001

Kulak, Daryl and Eamonn Guiney, *Use Cases: Requirements in Context*, Addison-Wesley, 2000

Lakoff, George and Mark Johnson, *Metaphors We Live By*, Chicago Press, 1980

Maiden, Neil, S. Minocha, K. Manning, and M. Ryan, *CREWS-SAVRE: Systematic Scenario Generation and Use*, Proc. ICRE'98, pp 148-155, 1998

Mills, Donald, SRE Mailing List postings (sre@ics.mq.edu.au), August 2001

Pinker, Steven, *How the Mind Works*, Penguin, 1997

Potts, Colin, Kenji Takahashi, and Annie Anton, *'Inquiry-based Requirements Analysis'*, IEEE Software, Volume 11, Number 2, pp 21-32, 1994

Potts, Colin, *Metaphors of Intent*, RE'01, 31-38, 2001

Reeves, Byron and Clifford Nass, *The Media Equation, how people treat computers, television, and*

new media like real people and places, Cambridge, 1996

Robertson, Suzanne and James Robertson, *Mastering the Requirements Process*, Addison-Wesley 1999

Scenario Plus, website (free Use/Misuse Case toolkit), http://www.scenarioplus.org.uk 2002

Simon, Herbert, *The Sciences of the Artificial*, 3rd Edition, MIT Press, 1996

Sindre, Guttorm and Andreas L. Opdahl, *Eliciting Security Requirements by Misuse Cases*, Proc. TOOLS Pacific 2000, pp 120-131, 20-23 November 2000

Sindre, Guttorm and Andreas L. Opdahl, *Templates for Misuse Case Description*, Proc. 7th Intl Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ'2001), Interlaken,
Switzerland, 4-5 June 2001

Stevens, Richard, Peter Brook, Ken Jackson, and Stuart Arnold, *Systems Engineering: coping with complexity*, Prentice-Hall, London, 1998

Swartout, William and Balzer, Robert, *On the Inevitable Intertwining of Specification and Implementation*, Communications of the ACM, Volume 25, Number 7, pp 438-440, July 1982