Misuse Cases

Use Cases with Hostile Intent

Ian Alexander

A version of this article appeared in IEEE Software, January 2003

Humans have analyzed negative scenarios ever since they first sat around Ice Age campfires debating the dangers of catching a woolly rhinoceros: *'What it turns and charges us before it falls into the pit?'* A more recent scenario is *'What if the hackers launch a denial-of-service attack?'*. Modern systems engineers can employ a misuse case–the negative form of a use case–to document and analyze such scenarios[1-3]. A Misuse Case is simply a Use Case from the point of view of an Actor hostile to the system under design. Misuse Cases turn out to have many possible applications, and to interact with Use Cases in interesting and helpful ways.

Eliciting Security Requirements

Security Requirements exist because people and agents that they create (such as computer viruses) pose real threats to systems. Security differs from all other specification areas as someone is deliberately threatening to break the system. Employing Use and Misuse Cases to model and analyze scenarios in systems under design can improve security by helping to mitigate threats.

Some misuse cases occur in highly specific situations, whereas others continually threaten systems. For instance, a car is most likely to be stolen when parked and unattended; whereas a Web server might suffer a denial-of-service attack at any time.

You can develop misuse and use cases recursively, going from system to subsystem levels or lower as necessary. Lower-level cases can highlight aspects not considered at higher levels, possibly forcing another analysis. The approach offers rich possibilities for exploring, understanding, and validating the requirements in any direction. Drawing the agents and misuse cases explicitly helps focus attention on the elements of the

scenario.

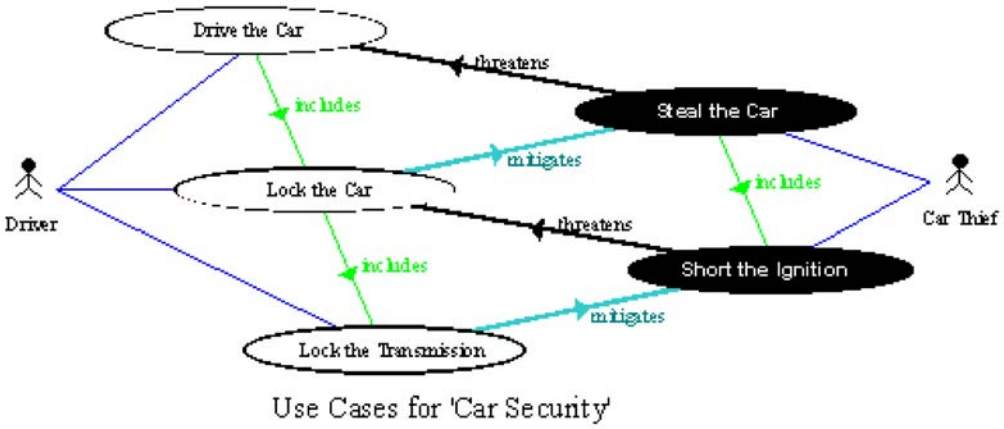


Figure 1. Use/misuse-case diagram of car security requirements.
Use-case elements appear on the left; the misuse cases are on the right.

Let's compare Figure 1 to games such as chess or Go. A team's best move consists of thinking ahead to the other team's best move and acting to block it. In Figure 1, 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 to authenticate the driver – is required. In this way, what starts out as an apparently simple hardware-only design may call for software subsystems. Threats to e-commerce and other commercial systems can be more complex, but may be analysed in the same way.

Figure 1 also shows that threat and mitigation form a balanced zigzag pattern of play and counterplay. This "game" can be reflected in an inquiry cycle style of development. 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 constraints (e.g. cost, weight, size, development schedule).

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 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.

For example, drivers often do not lock their cars when they stop and leave their vehicles for short periods.

They may even leave their keys in the ignition and the engine running, presenting thieves with excellent if brief opportunities. Can designers protect against this sort of misuse? There are plainly more cases to consider than those diagrammed above. Even a regular Use Case such as 'Stop at Traffic Signal' presents an opportunity for theft.

Safety Requirements from Failure Cases

Misuse Cases are not limited to eliciting Security Requirements, or threats from human agents.

Karen Allenby and Tim Kelly describe a method for eliciting and analysing functional safety requirements for aero-engines using what they call 'use cases' [Allenby & Kelly 2001]. They are of course well aware of the range of conventional hazard analysis techniques, but observe that functional hazards should be intimately derived from system requirements: it is inappropriate for safety engineers to go away and invent functions by themselves. Therefore they perceive a need for a method of deriving hazards from known system functions, proposing use cases for this purpose. 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 a recursive decomposition. However, since their 'use cases' describe potentially catastrophic failures and their effects, it would seem reasonable to follow Sindre & Opdahl and explicitly call them Misuse Cases. 'Failure Cases' is another suitable name.

In the case of safety requirements, there is not necessarily 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, drivers may lose control of their cars if the road is covered in ice or wet leaves. It can be advantageous to anthropomorphize the weather as an agent 'intending' to make the car skid (see diagram below for some simple examples). This uses the force of an easily-understood metaphor to emphasize the requirement for control in different weather conditions. Human language is always metaphoric, so it may be wise to express requirements in that familiar way [Potts 2001].

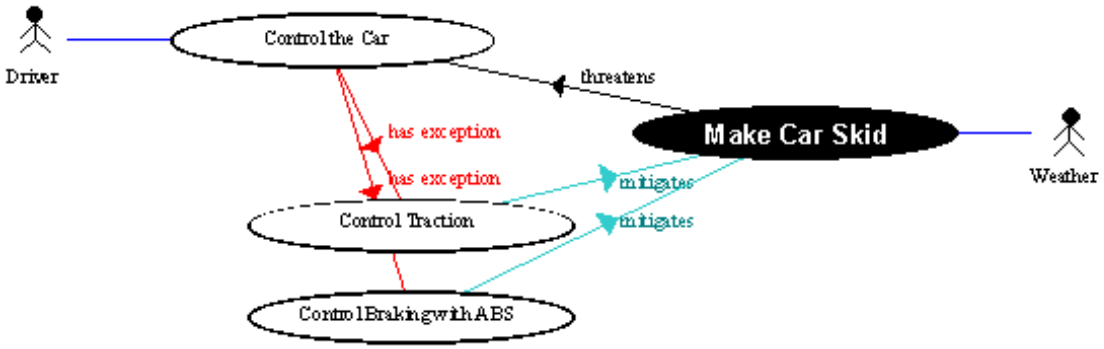


Figure 2. Eliciting and analyzing car safety requirements through use and misuse cases. 'Weather' is the negative agent.

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. 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 Scenario Plus.

As with Security requirements, Safety requirements may be developed recursively, going from system to subsystem levels and lower as necessary. Again, bottom-up and middle-out working remain possible. The explicit presence of Misuse Cases should make validation of safety requirements by domain experts easier and more accurate.

Threats to safety are traditionally evaluated in Failure Mode Effects Analysis (FMEA) and related techniques. Possible causes are related to possible effects, assumptions and measurements are made, and probabilities are calculated, leading to an assertion that the system is sufficiently safe. However, the analysis depends on the accurate identification of possible failure modes – no work is done on unimagined failures.

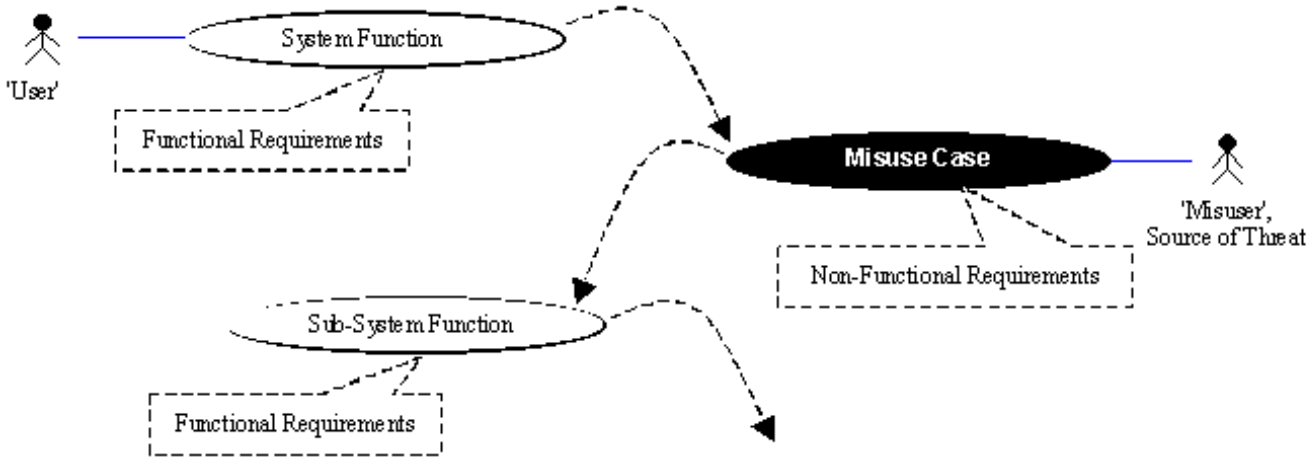
Therefore, any technique such as Misuse Case Analysis that seeks to identify possible causes of system failure can contribute to the safety of systems. Misuse Cases can feed FMEA with plausible threats to systems.

Interplay of Design, Functional and Non-Functional Requirements

The examples given so far illustrate the interplay of Misuse Cases with system and subsystem functions. A Misuse Case is an action – albeit a hostile one – that can be thought of as a functional goal: e.g. Steal the Car. The Misuse Case points out the existence of a threat. Threats were traditionally handled by writing non-functional requirements and standards governing quality. For instance:

'The car shall be constructed to the intrusion resistance (quality) defined in STD-123-456 '.

Does this mean, then, that Use Cases define functions and Misuse Cases define non-functional requirements? Possibly, but only if you look at the situation with traditional sunglasses. There is another way of looking at the role of Misuse Cases. This is to observe (see diagram below) that the typical response to a serious-enough threat of misuse is for the designers to create a sub-system (such as a lock), whose function (preventing intrusion) is to mitigate that threat. In shorthand, you can say that the Misuse Case elicits the subsystem function.



Interplay of Use & Misuse Cases with Functional & Non-Functional Requirements

The situation is that both Use Cases and Misuse Cases can help with eliciting both functional requirements and non-functional ones, if you accept the distinction. However, the diagram makes clear that there is a dynamic interplay between different types of requirement, and that one person's non-functional attribute is

another person's distinct subsystem function.

Elicitation through	Use Case	Misuse Case
Functional Requirement	**Important Mechanism**	Useful, but indirect
Non-Functional Requirement	Possible, sometimes happens	**Important Mechanism**

Applicability of Use & Misuse Cases for Eliciting Different Types of Requirement

If so, can Use Cases cover all types of requirement, and even (as some have claimed) replace them? Let us consider how Use and Misuse Cases can help to elicit further types of requirement.

Eliciting '-ility' Requirements

Misuse Cases can help to document the types of non-functional or quality requirement that engineers often call the '-ilities', such as Reliability, Maintainability, Portability, Testability and so on. There is no universally agreed definition of what constitutes an -ility, but several organizations have useful classifications for their own domains. The threats to system integrity and satisfactory performance are real enough, though. Here are some examples.

1. **Reliability**

   Whereas security threats stem directly from genuinely hostile agents, Reliability requirements may be elicited and analysed as threats caused by agents that are not necessarily intelligent. Such agents include human error, storms, design errors (e.g. software bugs), and interference on telecommunication links. They can cause software crashes and other types of failure.

2. **Maintainability, Portability**

   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. These simple examples illustrate that, contrary to the view that Use Cases only discover functions, Use/Misuse Case analysis can be applied to many types of requirement.

3. **Other '-ilities'**

It is not difficult to think of Misuse Cases for other '-ilities'. For instance, Usability: Novice Operator becomes confused by the user interface. The approach can also be applied to 'hardware' aspects, for example for Storability and Transportability: Icy Weather and Rough Handling damage the delicate components.

Simplicity is a virtue: it indicates that fundamental situations capable of affecting many types of system can be involved. If conversely Misuse Cases could only be expressed in terms of complex calculus, that might indicate they had limited applicability. Their simplicity suggests that they are rather robust: they form strong arguments in favor of designing systems in particular ways.

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

As for whether this means you can just model Use Cases and forget writing non-functional requirements and constraints, the jury is out. Some things that are easy to describe in a few words – the system must be delivered in 2 years, the unit cost must be no more than $250, the Mean Time Between Breakdowns must be at least 10,000 hours of operation – are perhaps not ideally written in scenarios. My own feeling is that it is often helpful to think through scenarios to elicit constraints and non-functional requirements, but that it is then often appropriate to summarize them as statements.

Eliciting Exceptions

An exception is an undesired event that could cause a system to fail. An exception-handling Use Case describes how the system under design can respond to such an event to prevent a possibly catastrophic failure. The response may lead to the resumption of normal operations, or it may lead to a safe shutdown – such as the emergency stopping of a train that passes a danger signal.

Misuse Case analysis is one way to hunt down possible exceptions. Sometimes it is worth documenting Misuse Case scenarios in a little detail; other times, just the name of the Misuse Case may be enough to identify gaps in a system's requirements.

There is a clear relationship between the hostile Actors who initiate Misuse Cases, and Exception Classes. Exception Classes are simply named categories of exception, generic situations that cause systems to fail. Where there is a proven list of Exception Classes for a domain such as naval warfare, the list can be used to generate candidate exception scenarios, and hence can elicit requirements to prevent system failure. However, such lists exist in very few domains.

Exceptions can also be elicited with simple requirements templates, just like the '-ilities' discussed above. Good templates are helpful in eliciting and validating requirements simply because they remind us of questions to ask, e.g. 'Could there be any portability requirements here?'. To elicit Exceptions, one steps through all the scenarios in the Use Cases, asking 'Could anything go wrong here?' This is effective and general, but not guaranteed to find all possible exceptions.

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 with Misuse Cases is sometimes a more powerful technique than simply stepping through a template or thinking about exceptions, 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 'Who might want this to go wrong?' and 'What could they do to make this go wrong? ', the Misuse Case approach contributes to searching systematically for exceptions using the structure of the scenarios themselves as a guide, and with a more specific intent than plain search for exceptions.

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. There is an obvious parallel here with Cost/Benefit analysis.

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

I find both templates and scenario-directed search for exceptions useful in requirements elicitation. Misuse Cases offer an additional way towards that holy grail, the 'complete' set of requirements.

Test Cases

Rather little needs to be said about the possibilities of Misuse Case Analysis for eliciting Test Cases. Evidently any scenario can lead to a test case. Good testing goes beyond happy-day scenarios to explore boundary conditions and exceptions.

Misuse Cases can clearly help to identify exceptions and failure modes. Any of these may be considered worth testing or verifying by other means. The habit of thinking out negative scenarios is arguably an essential skill for the test engineer.

Products of Use/Misuse Case analysis that can contribute to effective test planning include:

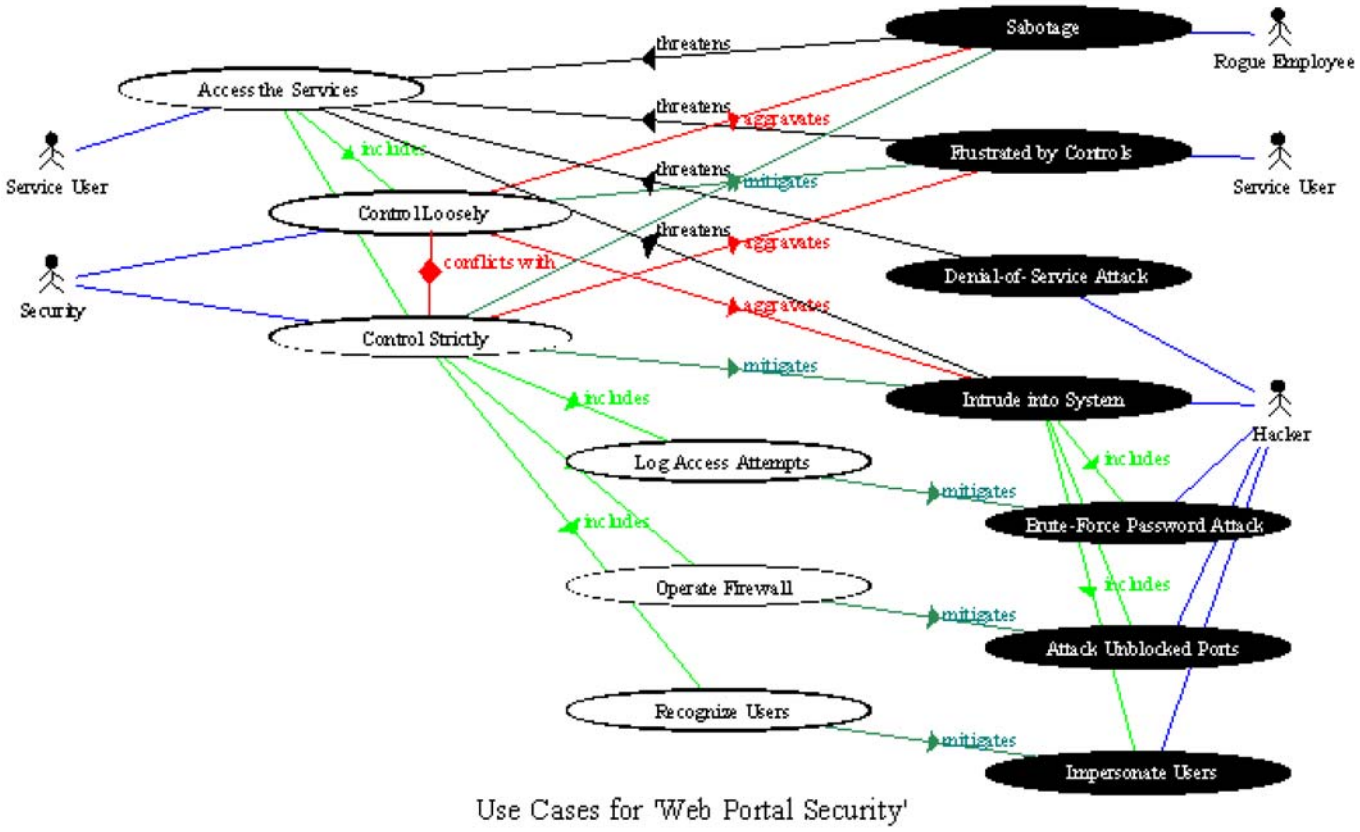
- Specific Failure Modes (especially useful for real-time, embedded, and safety-related systems)
- Security Threats (especially useful for distributed commercial and government systems)
- Exception-Handling Scenarios (always useful, often directly translating to test scripts).

A test engineer could view Misuse Cases as existing purely to ensure better system testing. A quality engineer could equally well argue that their purpose was to improve the quality of delivered systems.

Design Trade-Offs with Misuse Cases

An important element of system design is to satisfy the often conflicting demands placed on a system by its users. The situation is complicated by the fact that each design choice opens up new possibilities for both use and misuse. Designers must therefore trade off one option against another.

For example, a top-level goal for a web portal is for service users to be able to access the provided services. This is threatened by many different possible assaults on security, from sabotage by rogue employees through to sophisticated attacks by hackers. Use of the system is also threatened by security itself, if it is so strict that it leads lawful users to become frustrated and seek alternative services elsewhere. Loose controls are more comfortable for such users, but invite misuse.



Conflict Analysis builds upon Use/Misuse Case Modelling with additional relationships 'aggravates' and 'conflicts with'

Once the threats, mitigations, and possible conflicts between design options are identified, designers can make informed choices in the light of the system's goals and requirements. The Misuse Cases, their relationships, and the Use Cases that ultimately are not implemented (such as, say, loose security control) form part of the justification for the system's design. They could all be discarded, but only at the risk of repeating the same trade-off arguments at the next upgrade of the system.

Misuse Cases thus have a definite role to play during system design, and indeed in addressing design issues and trade-offs during in-service operations and maintenance.

Putting Use & Misuse Cases to Work

Engineers are starting to look at Use Cases for a range of purposes in systems of many kinds. I and my colleagues at DaimlerChrysler are investigating the appropriate forms of use cases to assist the 'recycling' of requirements for control software in cars [Alexander 2001, Alexander and Zink 2002]. Use Cases can assist the development not only of software, which is the domain addressed by popular accounts of use cases [e.g. Cockburn 2001, Kulak & Guiney 2000], but hardware and interfaces of all kinds. A large system composed of many subsystems, such as a passenger car, must be analysed in successively greater detail in subsystem models to cope with the complexity of the functionality. I believe that Use Case models of different kinds can be very helpful in expressing the purpose of system and subsystem features to different audiences.

There are good reasons for expecting that Use Cases should help with the elicitation, validation, and reuse of requirements. They should also help to guide design, to identify possible failure modes, and to generate test cases. Operational scenarios have indeed long been used for some of these purposes.

Engineers can apply Use and Misuse Cases at any level from whole systems down to individual components. I believe this approach dovetails well with both the recursive decomposition inherent in the SE life-cycle, and with participative, inquiry cycle approaches that bring users into the development team. Use and Misuse Cases help users and engineers communicate about development issues.

For example, a project developing software to control audio entertainment and safety announcements in a car needs to consider not just the driver's desire to be entertained – a basic Use Case – but also situations such as when a traffic announcement may override entertainment. Indeed, a safety announcement may override both entertainment and traffic, so the software system has to consider interactions between a range of subsystems.

Cars increasingly rely on software to perform functions that used to be considered as separate items of hardware. If a car has entirely separate radio, CD-player, and warning systems, then the radio and CD-player cannot interfere with the warning system – but music cannot be faded out to allow warnings to be heard clearly. Therefore car designers are moving towards integrating the control of all the devices that process information. The radio and CD-player are reduced to minimal hardware, and their control functions are implemented in software. Such integration permits desirable new behaviour such as fading audio during warnings, but also opens the way for undesirable interactions between subsystems.

Therefore, the entertainment subsystem cannot be developed in isolation; it must inherit some whole-car scenarios, and then develop its own more detailed Use and Misuse Cases to handle them. This is the Use Case equivalent of system decomposition and information hiding. Use/Misuse Case analysis can and I believe should contribute to each stage in system development, alongside other processes such as the identification of objects and the definition of messages to be passed between them.

Tool Support for Use/Misuse Cases

Use and Misuse Cases are fundamentally textual structures, and can certainly be handled as simple word-processed documents without special tools. The diagrams are also often quite simple, with a notation designed to be easy to draw.

However, a requirements tool specialized for Use Cases is probably amply justified to keep large numbers of cases organized and consistent through the life of a large project. A tool can automatically produce diagrams and metrics, check consistency, and guide requirements elicitation with a choice of more or less detailed templates.

I have produced such a toolkit for my own use, Scenario Plus [Scenario Plus 2001]. It is a free set of add-ons for DOORS that readers can download from the website. The diagrams in this article illustrate its graphical capabilities, though it should be emphasized that these represent only a part of its functionality – its templates to organize use case text, and its handling of links between cases are arguably more important.

To handle the interplay of Use and Misuse Cases, there are four combinations to consider, namely relationships to and from each kind of case:

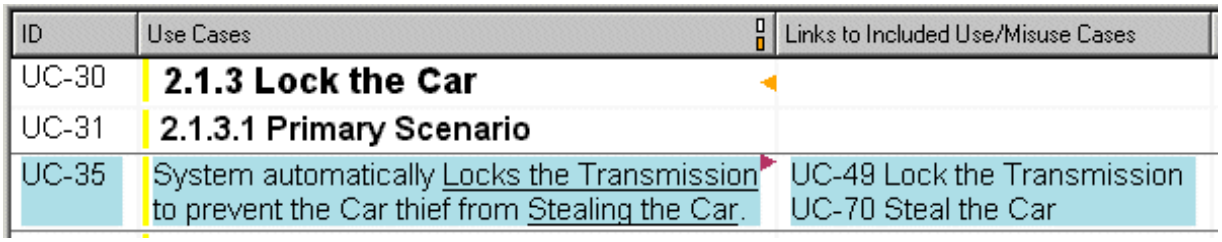
		Source Case	
	Case Type	**Use**	**Misuse**
Target Case	**Use**	*includes*	*threatens*
	Misuse	*mitigates*	*includes*

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. For example, a link

from a Misuse Case to a Use Case is assumed to be a threat, and is labelled '*threatens*'.

The Scenario Plus toolkit implements this mechanism to construct links. The requirements engineer names the target case within a scenario step in either the Primary Scenario or an Alternative Path of the source case, and underlines the name. The analysis tool then scans for underlined phrases, and attempts to match them with existing Use or Misuse Case names (their goals). For example, the tool fuzzily matches the phrase 'Stealing the Car' with the Misuse Case goal 'Steal the Car' (see screenshot). If no match is found, the tool asks the user if a new case should be created. The tool then links the cases according to the rule defined in the table above.



ID	Use Cases	Links to Included Use/Misuse 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 English, so that as Alistair Cockburn says, they can 'make the story shine through'. The created links are displayed, drawn on the diagram as relationships between Use/Misuse Cases, and can be navigated as usual in the requirements database.

Users can choose to show or hide Misuse Cases (along with their malign Agents and relationships) as desired. Misuse Case Agents are automatically drawn on the right of the diagram to keep them apart from ordinary Actors; the Misuse Cases themselves are drawn with inverted colours.

Getting Started with Misuse Cases

The best way to get started is to have a small informal workshop in which you and other stakeholders first ask whether there are any hostile agents (or inanimate forces of nature that can be treated as hostile agents) which might threaten your system. If there are, brainstorm a list of Misuse Cases for each such agent. If you already have Use Cases, go through these and search for ways in which they could be threatened by Misuse Cases. This may lead you to create more Misuse Cases, or it may lead to relationships between the Use and Misuse Cases you already have.

The next stage is to consider how to mitigate the Misuse Cases, should they arise. This may well lead to the creation of new subsystem functions in the form of Use Cases. It may be appropriate to consider how these might conflict, with a Conflict Analysis. It may also be appropriate to consider the costs and benefits of different design approaches, and hence to alternative possible subsystem functions.

Some Misuse Cases may require little documentation; others may be worth defining in more detail, with at least a fully-worked out Primary Scenario as for a Use Case. Subsystem Use Cases created in this process will need to be documented as usual, and then examined to see if they are in their turn susceptible to their own Misuse Cases. This may require several iterations.

Later in the project, you may want to consider all the Misuse Cases as candidate test cases to ensure the system under design behaves as expected.

Large projects with an established methodology may need to prepare Misuse Case guidelines to include with the rest of their standards. These should cover the application of Misuse Cases for requirements elicitation,

analysis, design trade-offs, and verification.

Summary

The interplay of Use and Misuse Cases during analysis can help engineers to elicit and organize requirements more effectively. Functional and Non-Functional requirements thus elicited similarly interplay with each other – and with design decisions – throughout the design process. Chances are that thinking about Misuse Cases will pick up numerous issues that could otherwise have caused systems to fail, or added time and expense to development projects.

Use and Misuse Case analysis offers a promising basis for eliciting and analysing requirements, making design trade-offs, and selecting test strategies for all types of system. It complements existing analysis, design, and verification practices.

References

Alexander, Ian, *Use/Misuse Case Analysis Elicits Non-Functional Requirements*, Computing & Control Engineering Journal, Vol 14, 1, pp 40-45, February 2003

Ian Alexander and Friedemann Kiedaisch, *Towards Recyclable System Requirements*, 9th IEEE Conf. and Workshop on Engineering of Computer-Based Systems, Lund, Sweden, April 8-10, 2002

Alexander, Ian and Thomas Zink, *Systems Engineering with Use Cases*, Computing & Control Engineering Journal, Vol 13, 6, pp 289-297, December 2002

Allenby, Karen and Tim Kelly, *Deriving Safety Requirements Using Scenarios*, Proc. 5th International Symposium on Requirements Engineering RE'01, pp 228-235, 2001

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

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

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

Potts, Colin, *Metaphors of Intent*, Proc. 5th International Symposium on Requirements Engineering (RE'01), pp 31-38, 2001

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

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

Other Papers