MISUSE CASES HELP TO ELICIT NON-FUNCTIONAL REQUIREMENTS

USE CASES DEFINE REQUIRED SOFTWARE FUNCTIONS USING STORIES. THERE IS CONTROVERSY ABOUT THEIR USE FOR SYSTEMS AND NON-FUNCTIONAL REQUIREMENTS. THEIR NEGATIVE FORM, MISUSE CASES, MAY BROADEN THEIR REACH.

by Ian Alexander

PANEL 1
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, reported in a companion contribution (December 2002 *CCEJ*, p.289) A simple approach that can be applied at any level (system, subsystem etc.) in *the life cycle was summarised in Fig. 2 of that* article. 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 organise 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.

Systems engineers are starting to look at use cases (Panel 1) for a range of purposes within the systems engineering life cycle. 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. Even if writers recommend iteration to make use case models more accurate and more detailed, they 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.

The greater range and the fact that embedded and real-time systems are often complex and safety-related means that systems engineers are often sceptical. This scepticism 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 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.

A negative form of use cases, misuse cases 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 (Panel 2).

In principle an approach combining use and misuse cases can be applied at any level from whole systems down through subsystems to individual equipment 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.

One interesting feature is that misuse cases seem naturally to lead to non-functional requirements

PANEL 2
USE AND MISUSE CASES

Ivar Jacobson introduced use cases in the context of his work on large telecommunication systems. 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. 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 neutralising. The misuse case and its hostile actor neatly encapsulate the idea of evaluating 'green'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.

(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 (Panel 3).

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 organising 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 NFRs.

SECURITY IS UNLIKE OTHER AREAS IN A SPECIFICATION AS SOMEONE IS DELIBERATELY TRYING TO BREAK THE SYSTEM

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 and Opdahl. The explicitly drawn malign agents and misuse cases (see Fig. 1) can help to focus attention on the battle against such negative agents and hence can improve system security.

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 Fig. 1). 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.

There is a productive analogy with game-playing here: the 'orange' team's best move consists of thinking ahead to the 'green' team's best move, and acting to block it. For example, if the misuse being threatened is the theft of a car, the orange player is the lawful driver and the green 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 Figure 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. Both use and misuse cases may include subsidiary cases of their own kind, but their relationships to cases of the →

PANEL 3
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 NFRs 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.

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.

TABLE 1 RULE GOVERNING CREATION OF RELATIONSHIPS BETWEEN USE AND MISUSE CASES

		source case	
		use	**misuse**
target case	**use**	*includes*	*threatens*
	misuse	*mitigates*	*includes*

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 crystallise and the system requirements can be stated more specifically.

Security threats are rarely neutralised 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 neutralising all possible threats is of course wishful thinking and cannot be stated as a requirement.

TOOL SUPPORT FOR CASE ANALYSIS

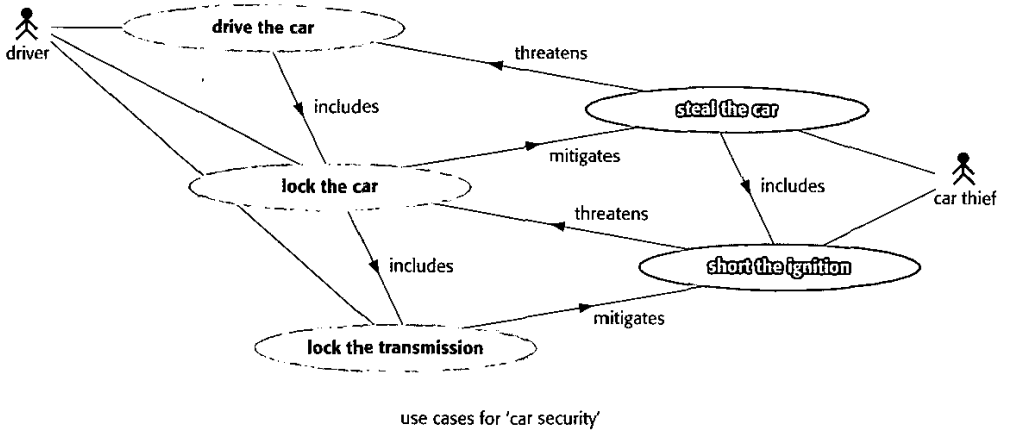
Considering the automatic handling of use cases and their interplay with misuse cases, there are thus four situations to consider, namely relationships to and from each kind of case (Table 1). This Table can be interpreted as a four-part rule governing the automatic creation of relationship types according 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 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 Fig. 2). 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 Table 1.

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 Fig. 2), 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 in Fig. 1; the misuse cases themselves are drawn (as Sindre and Opdahl suggest) on the right.

A refinement on Sindre and Opdahl is to permit misuse cases to exist at different levels relative to the

FIG. 1 USE/MISUSE CASE DIAGRAM TO ELICIT SECURITY REQUIREMENTS. HIGH LEVEL IS TO THE LEFT. USE CASES ARE DRAWN IN ORANGE. MISUSE CASES ARE DRAWN IN GREEN



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 in Fig. 1.

Mitigation can itself be subclassed into prevention, detection, and remedy. 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. It may be better therefore to document the precise relationships within the use cases themselves.

SAFETY REQUIREMENTS FROM FAILURE CASES

Allenby and Kelly have described a method for eliciting and analysing safety requirements for aero engines using what they call 'use cases'. 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 and Opdahl, and explicitly call Allenby and 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 Fig. 3). This uses the force of an easily understood metaphor to emphasise the requirement for control in different weather conditions.

The use of metaphor and anthropomorphism may appear colourful and even frivolous. However, human reasoning has evolved in a social context to permit sophisticated judgments 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 Fig. 3, 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 the Scenario Plus toolkit for DOORS.

As with security requirements, misuse and use cases may be developed in stages, 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 requirements by domain experts easier and more accurate.

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.

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 home-made 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.

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. 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.

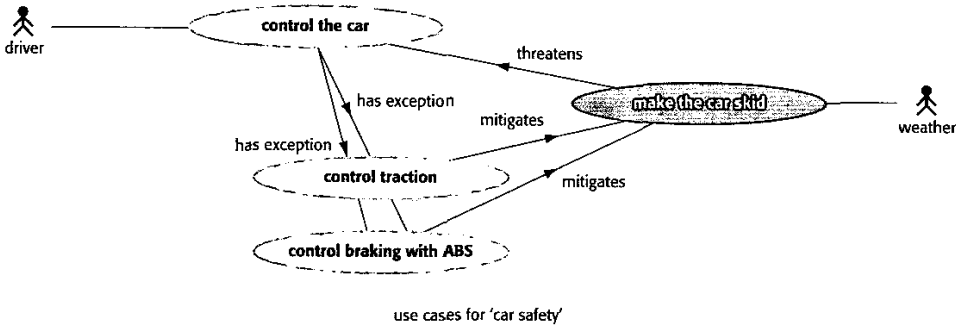
A system-level summary diagram can show such newly-created countermeasures use cases to indicate the role of subsystems in the design. Those use cases will naturally be drawn there at surface or low level. In the corresponding subsystem-level analyses, the same use cases will appear at overview or high 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 analysed in as much

Fig. 2 Automatic creation of links between misuse and use cases, by searching for underlined use case names with simple fuzzy matching

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

FIG. 3 USE/MISUSE CASE SUMMARY DIAGRAM FOR ELICITATION AND ANALYSIS OF SAFETY REQUIREMENTS. 'WEATHER' IS ANTHROPOMORPHISED AS A MALIGN AGENT



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 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:

- By inverting the problem from use to misuse, it opens a new avenue of exploration, helping to find requirements that might have been missed.
- By asking 'what can go wrong here?', it contributes to searching systematically for exceptions using the structure of the scenarios themselves as a guide.
- By being explicit about threats, it offers immediate justification for the search and indicates the priority of the requirements discovered.
- By personifying and anthropomorphising the threats, it adds the force of metaphor, applying the powerful human faculty of reasoning about people's intentions to requirements elicitation.
- By making elicitation into a game it both makes the search enjoyable and provides an algorithm for the search—orange thinks out green'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 (compare cost/benefit analysis).
- 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 trivialise 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. ■

Further information on this topic, including articles and references for further reading, can be found at the website: http://www.scenarioplus.org.uk.

Ian Alexander is an Independent Consultant. You can reach him at Ian.Alexander@scenarioplus.org.uk