A Look at Misuse Cases for Safety Concerns

Guttorm Sindre
Dept of Computer and Info Science, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway, guttors@idi.ntnu.no
WWW home page: http://www.idi.ntnu.no/~guttors

Abstract. Given the huge industrial take-up of UML, it has become less feasible to invent entirely new methods and modeling languages to address systems development challenges not covered by that language. Instead, the most fruitful way to go often seems to be to adapt UML to address such special challenges. In the security and safety domain, various such adaptations have been proposed. In this paper we look at misuse cases, originally proposed for security, with the purpose of investigating whether they are also useful for safety, and to what extent they can complement existing diagrammatic modeling techniques in the safety domain. Misuse cases is thus compared to several traditional techniques for safety analysis, such as fault trees, cause-consequence diagrams, HazOp, and FME(C)A, identifying strengths and weaknesses of either.

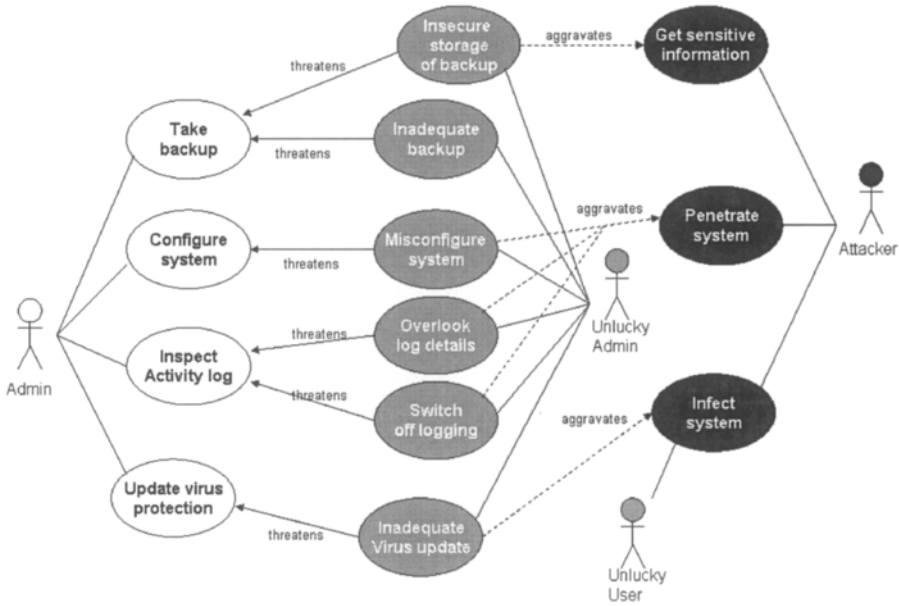
1 Introduction

As observed in [1] an increasing number of safety-critical systems are being fielded as IT penetrates more and more into the core operations of industry and society. Many problems with such systems stem from requirements defects. Although safety concerns may have been taken into account in the development of a system, unforeseen combinations of external events, system faults, and human failure may sometimes lead to disastrous effects [2]. There are many methods for safety analysis, some quite rigorous and other informal. Both have their advantages and disadvantages. The rigorous methods allow for formal analysis, perhaps automated. Informal methods may be better for creativity, e.g., imagining possible hazards, and for involving a diverse set of stakeholders in the discussion of safety concerns. Indeed, [3] points out both a better integration of formal and informal methods and a better integration of safety techniques with mainstream software engineering as important directions for improving the engineering of safety-critical software systems.

Please use the following format when citing this chapter:

Sindre, G., 2007, in IFIP International Federation for Information Processing, Volume 244, Situational Method Engineering: Fundamentals and Experiences, eds. Ralyté, J., Brinkkemper, S., Henderson-Sellers B., (Boston Springer), pp. 252-266.

One proposal for a technique towards the informal end of the spectrum, but which can easily be integrated with mainstream software engineering practices, is that of misuse cases [4]. These were originally proposed for looking at security threats, but in some case studies by Alexander [5, 6] the problems investigated were just as much concerned with safety hazards – the distinction being that security considers threats from malicious attackers, while safety considers hazards resulting from accidental human or system failure. The same notation can be used for safety-oriented as for security-oriented misuse cases, i.e., the misuse case uses an oval icon of the same shape as a use case icon, only inverted, and the actor performing the misuse case is similarly inverted compared to a normal use case actor. If addressing security and safety concerns in the same diagram, one might also distinguish between the two – as it might be of some importance whether something happens as a result of a malicious attack or because of an accident. One possible way of doing this is shown in Figure



1, using black nodes for security threats and grey nodes for safety hazards.

Fig. 1. Misuse case diagram with both safety and security threats

The diagram of Fig 1 does not show all the use cases of the system in question, only some of those relating to a system administrator, who in this case is responsible for backup, configuration, virus protection, and the inspection of activity logs. Middle and right it then shows safety-related misuse cases (grey) and security-related misuse cases (black). The difference between these would be that the grey ones result from human or system failure, while the black ones result from a malicious attack. Alternatively, all these might have been shown as black, not distinguishing the notation of various types of misuse cases. However, the explicit distinction between

safety and security makes the picture clearer, in particular demonstrating how human error on the system administrators side ("Unlucky Admin")[1], such as misconfiguring the system, may in turn increase the possibility for malicious attacks such as "Penetrate system". This is often the case: Many organizations have adequate firewalls, virus protection software etc., but vulnerabilities exist because the software is not used properly, necessary updates not done frequently enough, or similar.

It can be noted that diagrams quickly grow complex when trying to look at realistic examples and including both safety and security problems. In the rest of the paper we will therefore restrict ourselves to looking only at misuse cases for safety. Our research questions are as follows:

- *RQ1: Can misuse cases, both diagrammatic and textual, be applied in the safety domain?* (i.e., can they faithfully capture hazards and mitigations)
- *RQ2: Do misuse cases add value to other textual and diagrammatic representations traditionally used for safety hazard analysis, such as fault trees, cause-consequence diagrams, HazOp, and FME(C)A?* As observed in [2] the traditional techniques for hazard analysis do have limitations, so there is reason to believe that misuse cases could provide something useful, at least as a supplement to the existing techniques.

The rest of the paper is structured as follows: In section 2 we look at misuse cases vs. fault trees, in section 3 we compare them with cause-consequence analysis, in section 4 with HazOp, and in section 5 with FME(C)A. Section 5 provides comparison with related work, and then section 6 contains a discussion and conclusion to the paper.

2 Misuse Case Diagrams vs. Fault Trees

Fault tree analysis is a top-down method primarily meant to analyze the causes of hazards rather than identifying the hazards. From a top event (rectangle) which captures some harm that should be avoided in the system, decomposition into various possible causes for this continues until basic events are reached (circles). These basic events are those that really happen in the system, while the intermediate rectangles are pseudoevents (collective sets of basic events). Figure 2 shows an example of a fault tree for a Patient Monitoring System (PMS) adapted from [2], the top event is that wrong / inadequate treatment is given to the patient. This is then decomposed until basic events are reached. The gates with concave bottoms are OR-gates, i.e., the top event can occur if either of the two pseudo-events on level 2 happen, while gates with flat bottoms are AND-gates, i.e., for vital signs not to be reported, it is necessary that both the basic events below occur: the sensor failing and the nurse failing. We use a textbook fault tree rather than a self-developed one, to pose a less biased challenge for misuse cases. With a home-made example, there would be a greater risk that we either consciously or unconsciously made up one that could be represented elegantly with misuse cases.

[1] Of course, the Admin might not only be "unlucky" but possibly also incompetent, overworked or something else. The chosen phrasing, however, appears more sympathetic, and it must be remembered that in concrete projects the system admin in question will be one important stakeholder in the analysis.

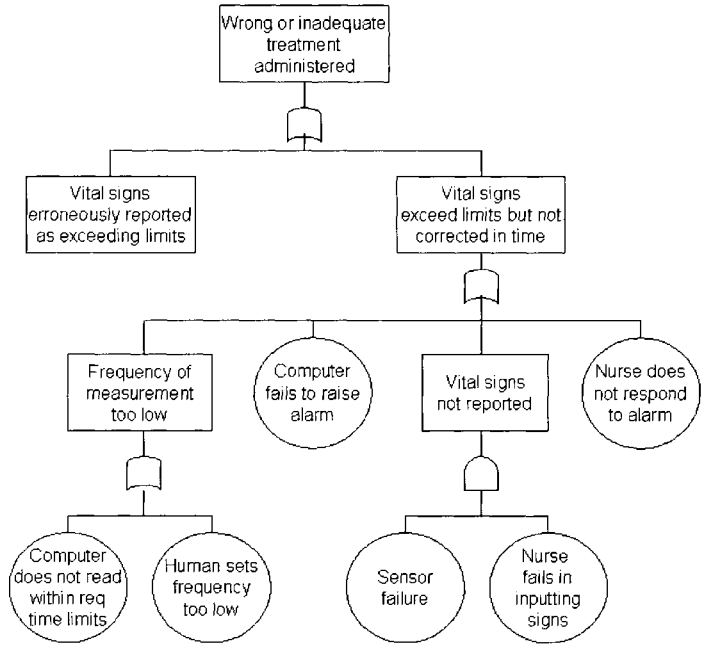


Fig. 2. Partial fault tree for Patient Monitoring System (PMS), adapted from [2]

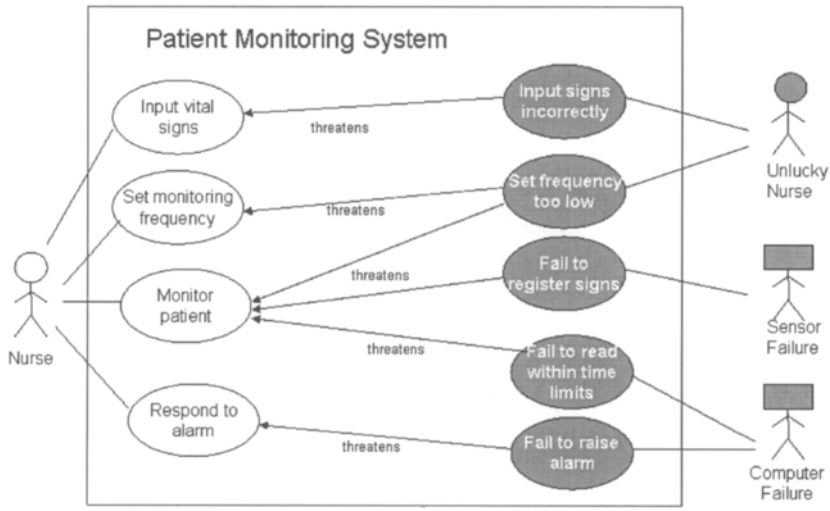


Fig. 3. First attempt at a MUC diagram for the PMS

A first attempt to represent the same problem domain with misuse cases is shown in figure 3, taking the PMS as the system boundary. Moreover, apart from standard UML diagram notation, we follow the recommendation of [7] that automated actors (e.g., computer systems, sensors etc.) be shown with square heads so as not to confuse them with humans. Nevertheless, there are several awkward things about this diagram:

- The top level hazard of the fault tree is not at all depicted. This is because "wrong treatment" is something that happens between the nurse and the patient. Use cases (and thus also misuse cases) concentrate on what takes place at the system boundary, and when the PMS is chosen as the system, the diagram therefore shows what the nurse is supposed to do in the PMS, but not what she is supposed to do with the patient.
- The diagram fails to show the hierarchy and AND/OR-relationships.

In the light of this example, misuse case diagrams mostly seem to have weaknesses relative to fault trees, but an interesting possibility is to put the patient at the center of the analysis. This yields a quite different misuse case diagram, shown in Figure 4. Now it is straightforward to represent not only the top level hazard of giving wrong treatment, but also the primary use case "Give treatment". This better discloses the motivation for the PMS in the first place: a nurse with no automated help would easily overlook emergency situations. A possible advantage of misuse case diagrams vs. fault trees is that while fault trees only show what can go wrong, misuse case diagrams show what is supposed to be done *together* with what can go wrong and can also indicate how use cases might mitigate the hazards. On the other hand, the diagram of figure 4 fails to show the interaction between the Nurse and the PMS. Combining the two views yields Figure 5. This would soon become too complex if taking more hazards into account, so it is most practical to develop diagrams looking at one system boundary at a time.

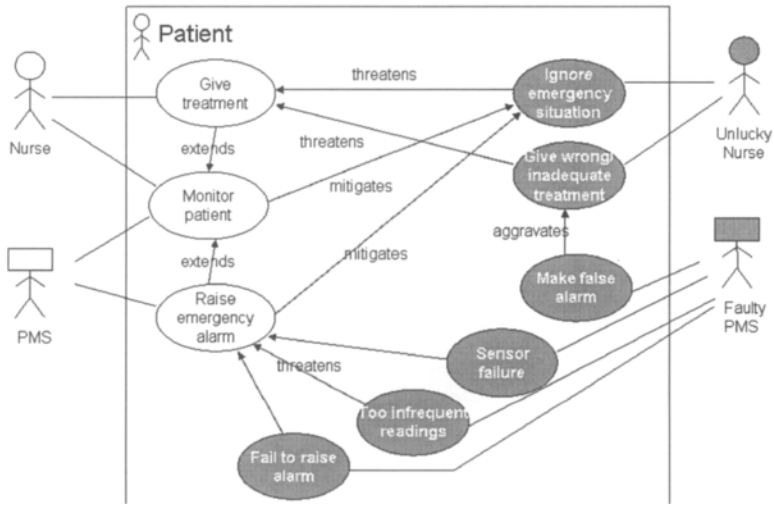


Fig. 4. New misuse case diagram, now with the Patient as system boundary

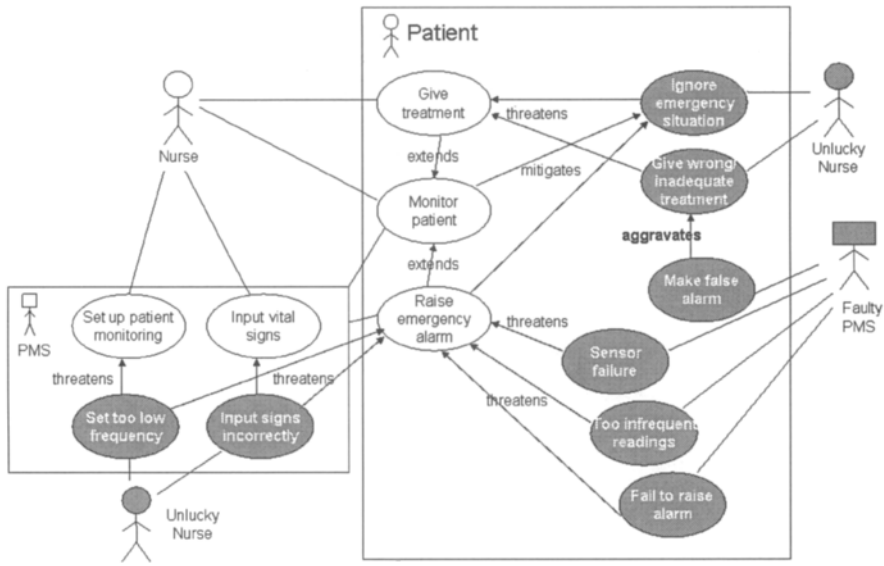


Fig. 5. Combining the two perspectives of Figure 3 and 4

3 Misuse Case Diagrams vs. Cause-Consequence Diagrams

Figure 6 shows an example cause-consequence diagram for a boiler system, again taken from [7]. Such diagrams are focused on a critical event – in this case "Pressure too high" – and look both at causes of the event (backward search) and results of the event (forward search). In the example, the cause is "Uncontrolled action", while the final harm if everything fails is "Explosion". Hence, the reading direction is from top to bottom, implying both a timeline and causal development. In addition to the square events, the diagram shows conditional boxes (with Yes/No continuations), basic conditions (circles), and OR-connectors. There are also AND-connectors, as well as several other symbols available which are not used in this small example.

Figure 7 shows a misuse case diagram representation of the same system. In this case, it is quite straightforward to represent the same phenomena in the misuse case diagram with one single system boundary. Each representation has advantages and disadvantages. The CCD explicitly shows the timeline and causal relationship between various events, and also has explicit Yes/No choices. In the misuse case diagram causal relationships are implicit in "threatens" and "mitigates" relationships, but no so easy to spot quickly since there is no standard reading direction for such causal chains. In addition to the OR (shown), AND, and XOR nodes, there are several other kinds of nodes available in CCD's that would be difficult or impossible to capture in a misuse case diagram, such as time delays.

On the other hand, the CCD only shows events or conditions that directly affect the critical event. Misuse case diagrams can also easily include mitigations with an indirect effect – as exemplified by "Raise alarm" in Figure 8 – which will make it

less likely that the Operator fails to open Valve 2. With CCD's one would normally use several diagrams in this case. It would also be straightforward to introduce indirect negative actions in the misuse case diagram, for instance that the Failing Operator accidentally switches off computer auto-response for high pressure (not shown in diagram), this could then be another misuse case threatening "Open relief valve 1". Finally, as mentioned for the fault tree example, misuse cases again have the potential advantage of better showing the relationship between what actors are supposed to do, and what can fail.

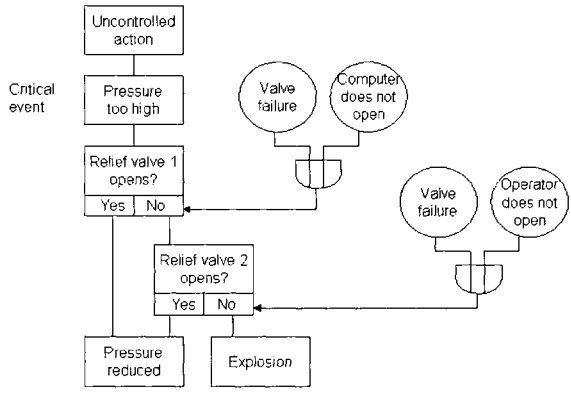


Fig. 6. Cause-consequence diagram (CCD) for a boiler system

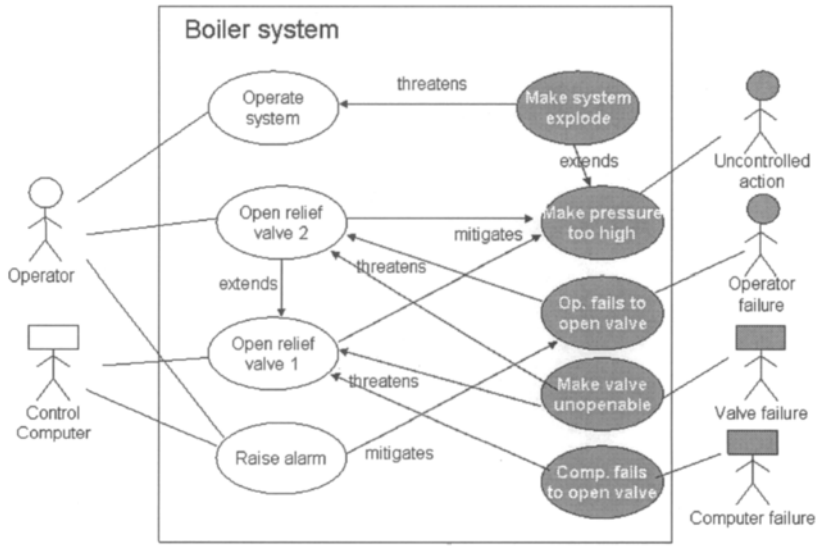


Fig. 7. Misuse case diagram for the boiler system

An overall observation here is that misuse case diagrams and cause-consequence diagrams have quite different purposes, the former meant for a high level overview of a system, while the latter is for a detailed exploration of specific chains of causation. Also, cause-consequence diagrams appear as a design level technique rather than requirements level, in that it must already be known which components the system consists of. For a requirements level misuse case diagram it would for instance be more appropriate to have a generic use case "Relieve pressure" than the two use cases assuming the existence of specific valves. Hence, rather than suggesting that one technique is better than the other here, a more natural question for further investigation would be whether they could be combined.

4 Textual Misuse Cases vs. HazOp

HazOp (Hazards and Operability Analysis) does not use diagrams but a textual representation, and would therefore be more relevant to compare with textual misuse cases. HazOp looks at the design of a system, e.g., a plant, and considers this system node by node, tentatively trying a number of guidewords for each node:

- NO: intended result not achieved
- MORE: more of a relevant physical property than there should be, e.g., pressure, voltage, temperature, flow, ...
- LESS: opposite of MORE
- AS WELL AS: something comes in addition to what was intended
- PART OF: only part of the intended result is achieved
- REVERSE, e.g., backflow instead of forward flow, closing valves instead of opening them
- OTHER THAN: the intended result is not achieved – as with NO – moreover, something completely different happens

A full HazOp requires that it is known exactly which nodes the system consists of, i.e., the design must be quite finished. This means that it is often too costly to change the design to avoid a hazard, so instead the outcome is to add protective measures. Many therefore prefer to do a preliminary HazOp considering early design sketches when it is still possible to change to a safer design. Still, even this preliminary HazOp is normally a design phase technique – although one could imagine looking at use cases as "nodes" for investigation rather than design components. The report produced by a HazOp will contain a number of table formed entries, an example of such an entry is shown in Table 1.

Table 1. Example HazOp entry, adapted from [7]

Guide word	Deviation	Possible causes	Possible consequences
NO	No flow	1. pump failure 2. suction filter blocked 3. isolation valve closed	1. overheating in heat exchanger 2. loss of feed to reactor

A direct attempt to translate the above into textual misuse cases is shown in Table 2. Clearly, the HazOp entry is more compact and easier to understand in this case, and not surprisingly, since misuse cases are not intended for describing systems at the design level, and especially not continuous physical systems like this one. Misuse cases share with use cases a primary suitability for systems where action is performed in discrete chunks. It can also be noticed that there is hardly any scenario path here.

Table 2. Textual misuse case reflecting the HazOp entry

Misuse Case Name: Pump failure causes flow loss
Basic path: bp–1. A pump fails. bp–2. Flow is lost.
Alternatives: a–1. Instead of pump failure, flow loss is caused by **a blocked suction filter.** a–2. Instead of pump failure, flow loss is caused by **a closed isolation valve.**
Hazards: **h1.** heat exchanger is overheated. **h2.** feed to the reactor is lost.

On the other hand, textual misuse cases can again do something else better than the traditional safety techniques, namely relate threats directly to the actions to be performed in the system. For instance, this could be done through the lightweight textual misuse case notation suggested in [4]. Table 3 shows an example where the normal use case is described in the two leftmost columns, following the format of an essential use case, as proposed by [7]. Then, an additional column is added on the right, describing the potential hazards. An advantage of this, compared to the HazOp entry format of Table 1, is that it becomes clear which specific step in an activity the hazard relates to. Also, the practice of looking for threats for each use case step can in itself contribute to creativity in brainstorming threats – as can be seen, 11 different hazards are identified only for this quite small use case. If one had been asked to imagine hazards just based on the use case name "provideEmergencyTreatment", not relating the hazards to the various steps, there is high chance that some of these hazards might have been overlooked.

Table 3: Example of lightweight misuse case notation including hazards per step

provideEmergencyTreatment		
Nurse intention	*PMS Response*	*Hazards*
Stay on guard		Nurse falls asleep
	Alarm nurse of emergency condition	PMS fails to raise alarm Alarm not noticed
Acknowledge alarm		Nurse fails to take action Nurse busy with other alarm
	Show patient and location	Wrong patient shown Wrong location shown
Administer treatment		Wrong treatment given Treatment to wrong patient
	Condition normalizes	Patient status incorrectly displayed
Log treatment		Erroneous logging, misinforming next shift

Clearly, HazOp and misuse cases thus complement each other, at least for systems where it is natural to apply use cases in the first place. A further possibility to consider would be if they could be combined more systematically, for instance whether HazOp guidewords could be applied for each use case step. When thinking about physical variables such as temperature, voltage and flow, it is quite clear what MORE, LESS, REVERSE etc. would mean, but this could be fuzzier when talking about actions involving information and human interaction, such as use case steps. For instance, looking at the sentence "The nurse administers the necessary treatment to the patient", we could have:

- NO: nurse fails to administer treatment
- MORE: administers too much of the treatment, e.g., giving an overdose of medication, or using too high voltage at defibrillation
- LESS: administers too little of the treatment.
- AS WELL AS: does something else in addition to the needed treatment, e.g., administration of another drug, too
- PART OF: gives only one of several treatments that were needed in conjunction. Another guideword that is sometimes used for time dependent systems, and which would clearly be of relevance in this example, is TOO LATE.
- REVERSE: not always applicable, e.g., cannot do the reverse of defibrillation, but might be relevant in some cases, e.g., decreasing rather than increasing the intensity of a drip
- OTHER THAN: meant to cover "something completely different", i.e., this should be wider than just the combination NO + AS WELL AS. Looking at the starting sentence, several OTHER THAN's could be imagined, e.g.,

OTHER SUBJECT: somebody else than the authorized nurse treats the patient, OTHER VERB: nurse does something else to the patient than giving treatment, OTHER DIRECT OBJECT: nurse gives something else to the patient than the treatment, OTHER INDIRECT OBJECT: nurse gives the treatment to another patient than the one intended.

As can be seen, not all of the hazards above are equally relevant in the given case, and for the latter bullet some seem partly overlapping. Yet, such a combination of use cases and guidewords may inspire the analysts to consider hazards that would otherwise have been overlooked, and these could then be captured in use cases like the one shown in Table 3.

5 Textual misuse cases vs. FME(C)A

Failure Modes and Effects Analysis (FMEA) uses forward search where the initiating events are failures of individual components (i.e., failure modes), then deriving from this what harm this could possibly lead to on a higher level (i.e., effects). Failure Modes, Effects, and Criticality Analysis (FMECA) is an extension of the same technique, where the criticality of each effect is analyzed in more detail. Table 4 shows a sample FMECA for a missile system, again adapted from [2].

Table 4: Sample FMECA for a missile system, from [2]

Failure Modes and Effects Criticality Analysis

Subsystem: _____ Prepared by: _____ Date: _____

Item	Failure Modes	Cause of Failure	Possible Effects	P	Level	Mitigation
Motor case	Rupture	a. poor work b.defective materials c.transport damage d.handling damage e.overpressurization	Destruction of missile	0.0006	Critical	Closely control manufacturing ...

This is difficult to translate to a misuse case, since "The missile's motor case ruptures" is a single event, not a use case style step by step action. Similar to HazOp the focus is also on design (and not on the requirements level) – one needs to be able to list all the components in the system and then perform FME(C)A on each of them. Still, the previous section suggested that a combination of misuse cases and HazOp might be useful, so the same might be the case for FME(C)A and misuse cases. Indeed, there already exists a proposal combining FMEA and use cases [8], but then looking at the use case as a whole when generating ideas for possible threats, and considering the use case as a "node" for investigation rather than looking at design components as nodes. A possible extension of this idea could be to look at threats per step and document the results in a textual misuse case similar to that of Table 3, or

similar to Table 2 if more details are needed. For each use case step one could then ask:

- What are the possible causes for failure of this step?
- What are the possible consequences of failure of this step?

This analysis could for instance prompt the inclusion of more alternative or exceptional paths in the use case, to address safety concerns and ensure that the use case as a whole might succeed even if one planned step fails. However, for space reasons the further exploration of this opportunity is beyond the scope of this paper.

6 Related Work

There are several techniques that seek to combine use cases or scenarios with safety analysis. [9] and [10] propose to apply use cases as input for performing safety analysis by means of safety engineering techniques such as Functional Hazard Assessment (FHA) [11] and HAZOP [12]. [13] proposes to formalize use cases to deal with the specification of fault-tolerant systems. In the CORAS project misuse cases were combined with other UML notations such as sequence diagrams, also specifically adapted for safety analysis [14].

Other adaptations of UML have also been proposed to address safety concerns, but then more on the design level. [15] extends UML specifically for safety, primarily achieved by profiles of packages, class and component diagrams. Safe-UML for modeling in the railway domain is proposed in [16]. Similar adaptations have been made for UML with respect to security in UMLsec [17] and SecureUML [18]. This work is different from the abovementioned in trying to compare the advantages and disadvantages of misuse cases relative to traditional safety analysis techniques.

7 Discussion and conclusions

In this paper we have mainly looked further at the possibility of applying misuse cases (both diagrams and text) for identifying safety hazards, which can be seen as a prerequisite for eliciting safety requirements. Several examples of safety-related misuse case diagrams and textual descriptions have been shown, comparing misuse cases with traditional safety techniques such as fault trees, cause-consequence diagrams, HAZOP, and FME(C)A. The examples show that these techniques, specially geared towards safety analysis, have many advantages versus misuse cases, and that it would certainly not be a good idea to throw away previous safety-analysis techniques and use misuse cases instead. In particular, it appears that misuse cases will be of little use in describing physical systems of continuous nature, where the accidents are sudden events rather than mistakes made in step-by-step processes. In short: if use cases are not suitable for the activity that goes on, neither will misuse cases be suitable for describing hazards of those activities.

Yet, on the other hand it has been indicated that misuse cases do also have something to offer for problems involving step-by-step activities. Better than the traditional safety techniques, misuse cases show the relationships between the

normal activities on the one side and faults and hazards on the other side. Also, it provides a simple intuitive notation which is well geared towards brainstorming, and if use cases or use case diagrams are anyway used in the project in question, the additional effort needed to apply misuse cases is small. Hence, it provides a good possibility for including safety concerns in the requirements phase of mainstream projects, in a similar way to what has previously been discussed for security. So, to answer the research questions:

RQ1: Can misuse cases, both diagrammatic and textual, be applied in the safety domain? Yes, but not necessarily for all problems. If the problem is dominated by continuous processes and physical concerns – and it is anyway not natural to apply use cases – then misuse cases are not likely to be suitable. On the other hand, for systems where activity is performed in a typical step-by-step manner appropriate for use cases, misuse cases can be suitable for early discussion of safety problems.

RQ2: Do misuse cases add value to other textual and diagrammatic representations traditionally used for safety hazard analysis...? Yes, but it is not all advantages – there are both pros and cons to misuse cases compared to other description formats. Hence, misuse cases cannot replace other techniques, but it could be useful in combination with other techniques. Typically, one could first use misuse cases for an early brainstorming of hazards, then follow up with other techniques such as fault trees, cause-consequence analysis, HAZOP, and FME(C)A. This also makes sense since these other techniques are primarily meant to be applied in the design stage, whereas misuse cases are intended for the requirements analysis stage.

Further investigation is of course needed to suggest in detail how misuse cases should be combined with other techniques. A particular weakness of the current paper is that all the examples looked at are small and simple. But at least, the examples were not invented by the author but taken from a textbook [2] which offered only one example for each of the techniques looked at. This means that the possibility for the author consciously or unconsciously to invent or pick examples that would be particularly fitting for misuse cases was removed, so at least the examples provide an unbiased challenge for misuse cases with examples that are representative of the safety field. But the most interesting topic for further work would of course be to investigate the application of misuse cases in real projects with significant safety concerns.

It can also be argued that safety issues might cause particular needs for situational method engineering [19-21], due to the following:

- Much variation in expertise: In some projects there are team members with much previous experience in safety analysis with traditional / heavyweight methods. In other projects this is lacking. And even in projects with high expertise, it may be necessary to use lightweight informal approaches at some stages to involve diverse groups of stakeholders with no competence in safety analysis.
- Variation in the degree of importance of the safety concerns. In some projects these are essential to every aspect of the software, while in others safety issues only apply to certain modules.
- Variation in the stage where safety analysis would take place. In some projects it might be natural to do some safety analysis very early (e.g.,

requirements elicitation stage), whereas in other projects participants might need to see some possible designs to be able to reason about safety issues.

One interesting topic for further work would therefore be to establish systematized criteria on the selection of safety modeling techniques such as those compared in this paper, based on task and stakeholder characteristics.

References

1. D.G. Firesmith, Engineering Safety Requirements, Safety Constraints, and Safety-Critical Requirements, *Journal of Object Technology*, 3 (3), 27-42 (2004).
2. N.G. Leveson, Safeware: System Safety and Computers (Addison-Wesley, Boston, 1995).
3. R.R. Lutz, Software Engineering for Safety: A Roadmap, in: The Future of Software Engineering, edited by A. Finkelstein (ACM Press, New York, 2000), pp. 213-226.
4. G. Sindre and A.L. Opdahl, Eliciting Security Requirements with Misuse Cases, *Requirements Engineering*, 10 (1), 34-44 (2005).
5. I.F. Alexander, Initial Industrial Experience of Misuse Cases in Trade-Off Analysis, in: 10th Anniversary IEEE Joint International Requirements Engineering Conference (RE'02), Essen, Germany, 9-13 Sep, edited by K. Pohl (IEEE, 2002).
6. I.F. Alexander, Misuse Cases, Use Cases with Hostile Intent, *IEEE Software*, 20 58-66 (2003).
7. 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, New York, 1999).
8. J. Zhou and T. Stålhane, A Framework for Early Robustness Assessment, in: 8th IASTED Conference on Software Engineering and Application, MIT, Cambridge, MA, 8-10 Nov, edited by M.H. Hamza (Acta Press, 2004).
9. K. Allenby and T. Kelly, Deriving Safety Requirements Using Scenarios, in: Fifth IEEE International Symposium on Requirements Engineering (RE'01), Toronto, Canada, edited by B. Nuseibeh, and S. Easterbrook (IEEE, 2001), pp. 228-235.
10. H.-K. Kim and Y.-K. Chung, Automatic Translation from Requirements Model into Use Cases Modeling on UML, in: Computational Science and Its Applications (ICCSA'05), Singapore, 9-12 May, Lecture Notes in Computer Science Vol. 3482, edited by O. Gervasi, M.L. Gavrilova, V. Kumar, A. Laganà, H.P. Lee, Y. Mun, D. Taniar, and C.J.K. Tan (Springer-Verlag, 2005), pp. 769-777.
11. SAE, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, Society of Automotive Engineers, Technical report, ARP4761, 1996 (unpublished).
12. F. Redmill, M. Chudleigh, and J. Catmur, System Safety: HAZOP and Software HAZOP (Wiley, Chichester, UK, 1999).
13. A. Ebnenasir, B.H.C. Cheng, and S. Konrad, Use Case-Based Modeling and Analysis of Failsafe Fault-Tolerance, in: 14th IEEE International Requirements Engineering Conference (RE'06), St.Louis, USA, 11-15 Sep, edited by M. Glinz (IEEE, 2006), pp. 343-344.
14. B.A. Gran, R. Fredriksen, and A.P.-J. Thunem, An Approach for Model-Based Risk Assessment, in: Computer Safety, Reliability, and Security, 23rd International Conference, SAFECOMP 2004, Potsdam, Germany, 21-24 Sep, Lecture Notes in Computer Science Vol. 3219, edited by M. Heisel, P. Liggesmeyer, and S. Wittmann (Springer, 2004), pp. 311-324.

15.J. Jürjens, Developing Safety-Critical Systems with UML, in: The Sixth International Conference on The Unified Modeling Language (UML'03), San Francisco, USA, 20-24 Oct, Lecture Notes on Computer Science Vol. 2863, edited by P. Stevens, J. Whittle, and G. Booch (Springer-Verlag, 2003), pp. 144-159.
16.K. Berkenkötter, U. Hannemann, and J. Peleska, HYBRIS - Efficient Specification and Analysis of Hybrid Systems - Part III: RCSD - A UML 2.0 Profile for the Railway Control System Domain (Draft Version), Univ. Bremen, Germany, 2006 (unpublished).
17.J. Jürjens, UMLsec: Extending UML for Secure Systems Development, in: The Unified Modeling Language, 5th International Conference (UML 2002), Dresden, Germany, Sep 30 - Oct 4, Lecture Notes in Computer Science Vol. 2460, edited by J. M. Jezequel, H. Haussmann, and S. Cook (Springer, 2002), pp. 412-425.
18.T. Lodderstedt, D. Basin, and J. Doser, SecureUML: A UML-Based Modeling Language for Model-Driven Security, in: The Unified Modeling Language, 5th International Conference (UML 2002), Dresden, Germany, Sep 30 - Oct 4, Lecture Notes in Computer Science Vol. 2460, edited by J.M. Jezequel, H. Haussmann, and S. Cook (Springer, 2002), pp. 426-441.
19.C. Rolland and N. Prakash, A proposal for context-specific method engineering, in: IFIP TC8, WG8.1/8.2 working conference on Method engineering: principles of method construction and tool support, Atlanta, edited by S. Brinkkemper, K. Lyytinen, and R.J. Welke (Chapman & Hall, 1996), pp. 191–208.
20.S. Brinkkemper, M. Saeki, and F. Harmsen, Assembly techniques for method engineering, in: 10th international conference on advanced information systems engineering (CAiSE'98), Pisa, Italy, Lecture Notes in Computer Science Vol. 1413, edited by B. Pernici, and C. Thanos (Springer, 1998).
21.I. Mirbel and J. Ralyté, Situational method engineering: combining assembly-based and roadmap-driven approaches, *Requirements Engineering*, 11 (1), 58-78 (2006).