3rd International Workshop on Requirements Engineering for High Assurance Systems (RHAS '04)

September 6, 2004
Kyoto, Japan

In conjunction with the 12th IEEE International Requirements Engineering Conference (RE '04)

Sponsored by the IEEE Computer Society Technical Council on Software Engineering (TCSE) and the Information Processing Society of Japan
# Table of Contents

**Welcome from the RHAS '04 Workshop Chairs** ................................................................. iii

**Program Committee and Workshop Chairs** ................................................................. v

**Agenda** .............................................................................................................................. vii

## Papers

Applying ISO/IEC 9126-1 Quality Model to Quality Requirements Engineering on Critical Software ............................................................................................................. 3
Motoei Azuma

Managing a System Safety Case in an Integrated Environment ................................. 11
Saeed Fararooy

A Taxonomy of Safety-Related Requirements ............................................................... 19
Donald G. Firesmith

Using Abuse Frames to Bound the Scope of Security Problems ............................... 29
Luncheng Lin, Bashar Nuseibeh, and Darrel Ince

Requirements Elicitation and Analysis Processes for Safety and Security Requirements ................................................................................................................................. 35
Nancy R. Mead

From System Safety Analysis to Software Specification ........................................... 41
Elena Troubitsyna
Welcome from the RHAS '04 Workshop Chairs

A high assurance system is a software-intensive system that must dependably deliver its services by exhibiting sufficient safety, security, survivability, reliability, robustness, and performance. Safety critical systems are those high assurance systems that can cause accidents resulting in significant harm to people, property, or the environment. For such systems, safety risks must be reduced to an acceptably low level. Thus, developers of such systems must eliminate or minimize hazards to avoid accidents and minimize the negative consequences of any accidents that do occur.

The goal of the RHAS '04 Workshop is to bring together in a set of small focused working groups researchers and practitioners from the fields of safety engineering and requirements engineering to exchange ideas and their experiences concerning the engineering of safety-related requirements.

Much research and development remains to be done on this important problem, and together researchers and practitioners need to identify and explore important subproblems and propose, formulate, and evaluate promising solutions. This third workshop on Requirements for High Assurance Systems is one of many forums that will allow new ideas to be proposed and discussed.

A set of topics has been identified for discussion at the workshop. Discussing these topics should clarify common assumptions and important issues. The accepted papers are each associated with at least one of the topics. The topics include the following:

- What types of safety-related requirements exist, and how do their differences affect the way they are elicited, analyzed, and specified?
- What are useful processes and techniques for engineering safety-related requirements?
- What kinds of tools are needed to support the engineering of safety-related requirements?

Additionally another topic to be selected by attendees will be discussed.

We thank the authors for their submissions, the members of the RHAS '04 program committee for their constructive reviews, and the organizers of the International Requirements Engineering Conference (RE '04) for providing a forum for the RHAS '04 Workshop. We also appreciate the support received from the Software Engineering Institute for the publication process. The support of Pamela D. Curtis, the SEI editor who prepared these proceedings, is especially appreciated.

Donald Firesmith, Co-Chair  
Software Engineering Institute  
Pittsburgh, PA

Nancy R. Mead, Co-Chair  
Software Engineering Institute  
Pittsburgh, PA
Program Committee

Motoei Azuma, Japan
Dan Berry, Canada
Ramesh Bharadwaj, U.S.A.
Anthony Hall, U.K.
Scott Hissam, U.S.
Tohru Kikuno, Japan
Robyn Lutz, U.S.
Bashar Nuseibeh, U.K.
Paul Tiplady, U.K.
Naoki Yonezaki, Japan

Workshop Chairs

Donald Firesmith
Software Engineering Institute
Carnegie Mellon University
e-mail: dgf@sei.cmu.edu

Nancy Mead
Software Engineering Institute
Carnegie Mellon University
e-mail: nrm@sei.cmu.edu
**Agenda**

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
<th>Chair(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00a – 9:15a</td>
<td>Welcome, Introduction, and Workshop Rules</td>
<td>D. Firesmith &amp; N. Mead</td>
</tr>
<tr>
<td>9:15a – 10:30a</td>
<td>Types of Safety-Related Requirements</td>
<td>D. Firesmith (Chair) &amp; M. Azuma</td>
</tr>
<tr>
<td>10:30a – 11:00a</td>
<td>---BREAK---</td>
<td></td>
</tr>
<tr>
<td>11:00a – 12:30p</td>
<td>Process for Engineering Safety-Related Requirements</td>
<td>N. Mead (Chair), L. Lin, &amp; E. Troubitsyna</td>
</tr>
<tr>
<td>12:30p – 2:00p</td>
<td>---LUNCH---</td>
<td></td>
</tr>
<tr>
<td>2:00p – 3:30p</td>
<td>Tools for Engineering Safety-Related Requirements</td>
<td>S. Fararooy (Chair)</td>
</tr>
<tr>
<td>3:30p – 4:00p</td>
<td>---BREAK---</td>
<td></td>
</tr>
<tr>
<td>4:00p – 4:30p</td>
<td>Another topic (to be decided by attendees) such as:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▫ Applicability of Standard Safety Techniques to RE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▫ Relationship Between Safety and Requirements Work Products</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▫ Relationship Between Safety and Requirements Teams</td>
<td></td>
</tr>
<tr>
<td>4:30p – 5:00p</td>
<td>Workshop Summary</td>
<td>D. Firesmith &amp; N. Mead</td>
</tr>
<tr>
<td>5:00p</td>
<td>Workshop Ends</td>
<td></td>
</tr>
</tbody>
</table>
ABSTRACT
In order to develop a software product for a critical system, specifying quality requirements is vitally important. Quality requirements should be defined based on various stakeholders’ needs. Software quality impacts the information system’s behavior, and the behavior impacts the behavior of the External-System that contains the information system. Safety is an issue of the External-System. A software product alone is harmless, because it can do nothing without computer hardware. However, any software quality characteristic, such as security and reliability, impacts the External-System’s safety. In this paper, conceptual models for quality requirements are presented. Then needs, requirements, and quality requirements are defined. Requirements for a Quality Requirements Engineering method are also stated. Then a method for Quality Requirements Engineering and associated specification is provided with a simple example.

Keywords
Quality requirements, ISO/IEC 9126, quality model, critical software

1. INTRODUCTION
In general, product quality has significant influence on developers, dealers and users of the product. For example, Y Food Company that caused large-scale food poisoning case was obliged to close its door. Software quality cannot be an exception. Especially, the quality of software products for mission critical systems has large impacts on its stakeholders, such as users and developers as well as the public.

In order to implement critical software quality, it is necessary that stakeholders’ needs are precisely analyzed and reflected in the requirements. One of the most important issues of requirements engineering is that the quality model for a target system and the quality model for the associated software cannot be the same. For example, in order to improve the safety of the target system, security, reliability, and usability of the software should be improved.

Though most of popular software requirements technologies support only functional requirements, it is extremely important that software quality requirements be clearly defined, especially for critical systems’ software [9, 10]. Quality requirements should be exhaustive for all quality characteristics within a quality model, because every quality characteristic of software product may have influence on system safety. Quality requirements should be also objective, accurate, and quantitative, and should also provide evaluation criteria [7].

A quality model is a very useful tool for quality requirement engineering as well as quality evaluation. ISO/IEC 9126-1 provides a software product quality model. It is intended to be used as a general purpose default standard quality model [3].

ISO/IEC JTC1/SC7/WG6 is developing ISO/IEC 25000 SQuaRE (Software Quality Requirements and Evaluation) series of international standards (IS), including new IS on Quality Requirements (25030), which is now in the 2nd CD stage [2]. WG6 decided to start to develop a new Quality Model (25010) as a revision of ISO/IEC 9126-1 Quality Model. WG6 is also planning to revise 9126-2, -3, and -4, External, Internal and Quality-In-Use Metrics respectively, as a part of the SQuaRE series of International Standards [4, 5, 6].

The purpose of this paper is to suggest an idea of how to use associated quality metrics for safety critical systems requirements engineering, to list issues, and to invite contributions of experts in order to make the SQuaRE series more useful. Contributions on metrics on safety and security are also expected.

The SQuaRE series themselves are general purpose. However they must be consistent with other communities’ standards, such as safety, security, reliability, dependability, and usability.

The concept and definitions of needs, requirements and quality requirements are stated in this paper. Then a method for defining quality requirements is proposed. The method is based on experiences but is not yet well validated.

2. CONCEPTUAL MODEL FOR QR
Figure 1 shows the relationships between system and software. An information system consists of computer and communication hardware devices and software products. An External-System consists of information systems, people, machines, buildings and other artifacts. Examples of External-Systems include business systems,
factory-automation, cars, aircrafts, and electric power generation plant.

A problem is defined as two states, i.e. the current states and the goal states of an External-System, its components, human factors, as well as its environment. Current states are effects of the current system’s behavior, and goal states are expected effects of the proposed system that is defined by requirements.

However, as the desirable states are not always obvious and may be different depending on each stakeholder’s position, the “problem” is redefined using goal states. When the desirable states are precisely defined as goal states and a system that is supposed to achieve the goal states are defined as requirements specifications, it is the proposed solution. Goal states should represent stakeholders’ needs.

Figure 2 shows the relationships between the current system, proposed system and realized system. Horizontal lines represent the transition of systems and software, and vertical lines represent cause and effect relationships. For example, current software is a cause of the current information system’s behavior.

3. CONCEPTS AND DEFINITIONS

Needs

Needs for a product are expectations of stakeholders for the effects of the product when it is actually operated, which means such action to the software product as development, distribution, release, installation, use and maintenance. In this context, stakeholders include developers, salesmen, system managers, users, end users, and maintainers. Stakeholders have their own needs for a product depending on their own positions.
“Needs” are categorized into stated needs and implied needs. Some needs are implied either because a stakeholder thinks it is too obvious to actually state the needs, or because no one is aware of the needs.

Some needs are contradictory. For example, a novice user wants a software product that is easy to learn and use; on the other hand, experienced engineers may want a product that is fast to use with many functions. Therefore, needs should be identified and selected for each Context-Of-Use. In other words, requirements should be derived from the stakeholders’ needs. Figure 3 shows the relationships between needs, requirements and design.

In this figure, an arrow means a relationship of “transform to” and “derived from”. For example, “External-System Requirements” are derived from “Selected & Specified Stakeholders’ Needs”, and “External-System Requirements” are transformed into “External-System Design”.

Requirements
In order to clarify the relationships between needs and requirements, “requirements” is defined in this paper as follows.

Requirements: Requirements are the external specification of specific needs that a product is expected to satisfy.

SWEBOK describes software requirements as follows [11].

“At its most basic, a software requirement is a property which must be exhibited in order to solve some problem in the real world. Hence, a software requirement is a property which must be exhibited by software developed or adapted to solve a particular problem.”

Functional requirements describe the functions that the software is to execute; for example, formatting some text or modulating a signal.

Non-functional requirements are the ones that act to constrain the solution. Non-functional requirements are sometimes known as constraints or quality requirements. They can be further classified according to whether they are performance requirements, maintainability requirements, safety requirements, reliability requirements, or one of many other types of software requirements.

System requirements are the requirements for the system as a whole. In a system containing software components, software requirements are derived from system requirements.

Specified requirements do not always satisfy selected and specified needs. Therefore it is necessary that the specified requirements be validated so that the proposed system will satisfy the needs at the earliest possible stage of software development lifecycle using, for example, prototyping.
Software Quality Requirements

Information System’s requirements and design should be transformed into software quality requirements, i.e. Functional Requirements and Quality Requirements (Figure 3). In order to clarify the concept of quality requirements, the following definitions are applied.

**Quality**: the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs [3]

**Quality model**: the set of characteristics and the relationships between them which provide the basis for specifying quality requirements and evaluating quality.

When software quality is measured and evaluated by attributes of the software product itself, the quality is named as internal quality.

ISO/IEC 9126-1 defines a quality model that should be used as the default quality model. The model defines three types of software quality: Quality-In-Use, External Quality, and Internal Quality. They are defined as follows.

**Quality-In-Use**: the user’s view of the quality of the software product when it is used in a specific environment and a specific Context-Of-Use.

Quality-In-Use measures the extent to which users can achieve their goals in a particular environment, rather than measuring the properties of the software itself. The concept of Quality-In-Use by this definition is close to the concept of users’ needs.

**External Quality**: the totality of characteristics of the software product from an external view. It is the quality when the software is executed, which is typically measured and evaluated while testing in a simulated environment with simulated data using external metrics.

**Internal quality**: the totality of characteristics of the software product from an internal view. Internal quality is measured and evaluated against the internal quality requirements.

The ISO model consists of six internal and external quality characteristics. It also defines four Quality-In-Use Characteristics, i.e. Effectiveness, Productivity, Safety, and Customer Satisfaction. As Customer Satisfaction usually reflects all quality properties, the author modified the model a little. The modified quality model is shown in Figure 4. Three Quality-In-Use Characteristics written in italic characters are new.

Safety, which is Quality-In-Use Characteristic, is defined in ISO/IEC 9126-1 as;

**Safety**: the capability of the software product to achieve acceptable levels of risk of harm to people, business, software, property or the environment in a specified Context-Of-Use.

A software product itself is completely safe, because it can do nothing. It does something only when it is executed as a part of an information system. An information system...
itself outputs only information, either correct or erroneous. Erroneous output information may affect the safety of the External-System. Every characteristic of external and internal software quality has some possibility of causing safety problem on the External-System. Therefore, internal quality does not include the safety characteristic in this quality model.

Based on these definitions, software quality requirements can be categorized into External Quality Requirements, Internal Quality Requirements, and Quality-In-Use Requirements. Each category of software quality requirements is defined as follows.

**External Quality Requirements** specify the required level of quality from the external view. They include requirements derived from user quality needs, including Quality-In-Use requirements. [3]

**Internal Quality Requirements** specify the level of required quality from the internal view of the product. Internal quality requirements are used to specify properties of interim products, including static and dynamic models, other documents and source code.

**Functional Requirements** are requirements for algorithms that transform input to output. The same input may cause different system behavior based on the state of the system.

Functional requirements and functionality requirements are considered to be different. Functionality is one of six Quality Characteristics that ISO/IEC 9126-1 defines. It is defined as;

**Functionality:** The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.

Therefore, while Functional Requirements define all functions that are necessary to satisfy selected and specified needs, Functionality Requirements provide decision criteria that contribute to deciding the priority of each function when the software product is used under specific condition, in other word, Context-Of-Use.

Probably, apart from Functional Requirements, Reliability Requirements is the most popular concept and frequently specified. It can be specified qualitatively using such measures as MTBF (Mean Time Between Failure) and MTTR (Mean Time To Repair). Software reliability is influential to safety of the External-System.

Usability Requirements should be stated by considering the users’ profile, such as experience, operational skill, eyesight, and taste. In the case of defining Usability Requirements for critical systems in which misjudgments and/or improper operation by a user may cause a serious disaster, special care should be taken for specifying Usability Requirements.

4. REQUIREMENTS FOR REQUIREMENTS ENGINEERING METHODOLOGIES

**Requirements for a needs analysis method**

Based on the concepts and definitions stated above, requirements for a needs analysis method are as follows.

(1) A method should be applicable to every type of stakeholder and should support solicitation not only of stated needs but also implied needs.
(2) A method should support selecting a need from alternatives or harmonizing contradictory needs.

(3) Selected needs should be specified as formally as possible, by using form sheets, screen form, or language such as XML.

Requirements for a Requirements Engineering Method
Based on the concepts and definitions stated above, requirements for a Requirements Engineering method are as follows.

(4) Requirements should be specified objectively and should provide criteria for validation.

(5) Requirements specifications should be easily understandable not only by requirements analysts but also other stakeholders, including software designers, users, and end users.

(6) Requirements specification should include not only functional requirements but also quality requirements, as well as constraints. Other information that influence the requirements, such as users' profile and the environments of the software, should also be stated.

Requirements for Quality Requirements Engineering Method
Based on the concepts, definitions, and requirements stated above, requirements for a Quality Requirements Engineering and Specification Method are as follows.

(7) Quality Requirements should be specified for all Quality Characteristics based on their criticality [8].

(8) Quality Requirements should be specified as formally and comprehensively as possible.

In this respect, use of a well designed form is helpful.

(9) Quality Requirements should be objective, and should provide measures and evaluation criteria.

(10) Quality Requirements should be reflected in the Functional requirements.

5. PROPOSED QR ENGINEERING METHOD

Outline of the Method
Quality requirements should be specified based on specified needs and functional requirements. The author proposed Quality-In-Use concept and related methodology at the 2nd International Conference on Software Quality [1]. The following is a Quality Requirements Engineering Method based on the same concept but improved it a little.

Recently, more attention has been focused on object oriented approaches than structured approaches. However, the structured approach has strength from the human cognitive capability aspect. Therefore, the proposed method combines the merits of both approaches. It consists of two activities, i.e. General Requirements Engineering and Detailed Requirements Engineering. It means that stakeholders can understand their own needs if they have an overview of the system.

General Requirements Engineering
The following is outline of the General Requirements Engineering method.

(1) State an outline of the target system in order to show it to the stakeholders and explain the purpose and outline of the target system for the purpose of soliciting their needs.

(2) Draw a first cut of the system overview diagram based on the outline statement. Use Case Diagrams and IDEF0 are useful for the purpose. However, as this diagram is a very rough image, it should evolve over time.

(3) Interview with major stakeholders and solicit their needs.

(4) Select and define the collected needs.

(5) List major functional requirements.

(6) List Actors.

(7) Analyze Overall Risks.

(8) List major quality requirements for each quality characteristic such as safety, reliability and usability. A quality model, e.g. ISO/IEC9126-1 should be used for this purpose.

(9) List required constraints and conditions, including total budget, delivery date, hardware and communication network environment and available human resources.

(10) Refine system into sub-systems and re-define outline statement. Refine system overview diagram and project description based on the defined sub-systems.

Detailed Requirements Engineering
Iterate the process explained in clause 5.2 for each sub-system more accurately and precisely.

(1) State an outline of each sub-system in order to solicit detailed needs in terms of Quality-In-Use Requirements.

(2) Draw a Use Case Diagram for Each Sub-system.

(3) Interview the major stakeholders, especially the major users and solicit their needs.

(4) Identify and Clarify Context-Of-Use (COU).

COU may be started with a Use Case Scenario, but it must have more information, which especially relates to quality requirements and should be more formal. Use a tool that supports describing COU and that converts the COU to XML.

COU should include but is not limited to users and their profiles, other actors, target tasks, methods of usage,
environment, frequency of use, potential risk, time constraints etc. The following is an example of a COU description.

Actor:

Name: user
ID: Sub-system A, User type 1
Profile:
   Age range: (~10, 11~20, 21~50,)
   Sex: (Male, Female, Both)
Experience: None
Skill of operation: None
Task:
   Name: Order entry
   ID: SSO-1
   Description: A user selects a product, type in quantity, select payment method....
Constraint:

Environment:
   Hardware:
   Operating System:
   Communication network:

(5) Analyze the Context-Of-Use and specify functional requirements for each Use Case.
(6) Select actors from COU and categorize Typical Users
(7) Analyze the Risks for Each Use Case and reflect them in the quality requirements
(8) Specify Quality-In-Use Requirements for each Use Case

The following is an example of a QIU specification.

Quality-In-Use Requirements
   Use case ID:
   QIU characteristic:
   Effectiveness requirements:
   Safety Requirements:
(9) Specify External Quality Requirements for each quality characteristic such as reliability and usability.
   External Quality Requirements
   Functionality:
   Security:
   Interoperability:
   Reliability:
   Usability:

(10) List required constraints and conditions, including total budget, delivery date, hardware and communication network environment and available human resources.
(11) Measure the QIU and Analyze the Results
(12) Refine the system outline statement, system overview diagram and project description.

6. ISSUES
Though experiences in an experimental scale at a university and companies with the method are promising, there are some issues for improvement. Examples include:

(1) Formalize the method: The method at this moment is not well defined. More effort is required for formalizing the method.
(2) Validation of the method by empirical studies: After the method is defined in a formal specification, the method should be distributed for beta test for the purpose of empirical validation.
(3) Methodologies from quality requirements to design: Some quality requirements should be reflected in the functional requirements. Some others may be reflected to the program architecture, structure and programming style. Guideline for this process should be developed.
(4) Support requirements change: Changes in requirements are always inevitable. At this stage of maturity, the method does not support requirements change. A method that predicts possibility of future changes and reflects it in the design should be developed.
(5) Quality measures: Requirements should be objective and quantitative. It also should provide evaluation criteria for delivery or acceptance. ISO/IEC JTC1/SC7 developed ISO/IEC 9124-2, -3 and -4 Quality metrics series technical report. JTC1/SC7 is also planning to revise the series for international standard as parts of SQuaRE series of international standard. However, in order to make it practical, more time and effort are needed [2] [3] [4] [5]. (Note: For the latest information, refer to http://www.jtc1-sc7.org/)
(6) Quality requirements standard: ISO/IEC JTC1/SC7 is developing a “Quality requirements” international standard as a part of SQuaRE series international standard. However, in order to make it practical, more time and effort are needed [2].
7. CONCLUSION
In this position paper, conceptual models for quality requirements are presented. Then definitions of needs, requirements and quality requirements are defined. Requirements for a method that support quality Requirements Engineering are also stated. Then a quality Requirements Engineering and specification methodology is provided with a simple example. While writing the paper, issues are identified. In order to find solutions for these issues, experts' opinion and further discussion are important. For this respect, the workshop is a good opportunity. The author believes that international standard can contribute for improving software quality for critical systems. Unfortunately both approaches are time consuming. However, we cannot wait developing and using critical systems until we find the best solution. Meanwhile, we are obliged to use better solutions existing candidates.

ACKNOWLEDGEMENTS
The author express acknowledgement to ISO/IEC JTC1/SC7/WG6 members for their dedicated contributions for developing ISO/IEC 9126 series and SQuaRE series of international standards. I also thank to reviewers who read carefully and send me very informative comments.

REFERENCES
1. Azuma, M., QUALITY IN USE; Its Concept, Metrics and Methodology, Proceedings 2WCSQ, 2000
ABSTRACT
This paper presents the challenges for development and maintenance of the safety case for electrical, electronic and programmable electronic systems (E/E/PES) used in safety-critical applications. A Remote Condition Monitoring (RCM) system is taken as a show case to demonstrate the elements of a safety case from concept, requirements and design through implementation, test, integration and transition to use. This reviews the techniques that may be used to manage safety issues at different phases of the project, namely:

a) Safety case data models such as Goal-Structuring Notation (GSN) to assist with the top-down planning of the safety goals, strategies, assumptions, models and context as well as arguments and evidence

b) Preliminary hazard analysis and the structure of a hazard log, its population and management
c) Compliance with safety regulations & standards
d) Full traceability and audit trail
e) Management information such as a summary safety risk classification matrix

Keywords
safety cases, hazard log, safety requirements, goal structuring notation, traceability

1 INTRODUCTION
When managing a complex system, at least one central shared database is indispensable for a team of developers and all other stakeholders. Safety is one important aspect of such a complex systems and is best addressed under the “fitness for purpose” paradigm. In other words: Is the system designed, developed, implemented, operated and maintained to be as safe as it needs to be? A recent study by the UK Health & Safety Executive (HSE) found that a greater volume of significant hazards were identified at the early conceptual and requirements phase of any project/product development. Yet, there is a knowledge gap between the safety engineering community and the systems engineering and requirements management world. The following problems with existing approaches based on status-quo were identified:

- Non-common processes – Parallel and disparate processes for managing safety as opposed to managing other system requirements;
- Reinventing the wheel each time a new safety case project commences;
- Non-integrated approach – Generally many obstacles in unifying engineering and acceptance team efforts and insufficient integration within each team (‘silo-working’ leading to ‘weakest-link’ syndrome);
- Manual traceability – The most common IT tools for safety case production remain MS Office (Word, Excel, …) resulting in difficulty of traceability between multitudes of separate generated documents;
- Lack of Control Tools – It was not possible to impose a credible technical management plan to monitor and review the safety case production process.

The approach promoted in this paper is one that attempts to overcome the above shortcomings. We have seen through dialogue with over 300-strong system safety and systems engineering community in the past year that the above problems are being recognised and several attempts are being made to bring an integrated approach to manage safety in synergy with other system features. In section 3, we introduce a case study of a novel distributed RCM system to demonstrate issues related to the development of a safety case. These include HAZard IDentification (HAZID, section 4), compliance with requirements in safety standards, regulations and best practice (section 5) and presentation of a structured case for safety with appropriate argumentation and supportive evidence (section 6). In section 7, we outline the features of an Integrated Safety Case Development Environment (ISCaDE), a commercial off-the-shelf software package, that we used to bring together all elements of the RCM system safety case. Benefits of the approach adopted and the integrated environment used are summarised in the conclusions.

2 RCM SYSTEMS
The Railwise RCM system under study here is a prototype distributed system based on fieldbus technology installed over existing safety-critical railway signalling equipment. It was developed as a collaborative industry-academia research programme at the University of Birmingham (UK).

The Railwise RCM system is designed specifically for the
purpose of providing warning of incipient failures in safety-critical electro-mechanical systems such as signalling equipment (point machines, level-crossing). It allows the prediction of faults by input analogue and digital signals into knowledge of the state of the health of equipment being monitored. The system is intended to be used in the railway industry for remote condition and event monitoring. It has the following functionality:

- Collects data from sensors attached to signalling asset (Equipment Being Monitored - EBM)
- Transmits data to a trackside PC (TSPC)
- Transmits data to Central Remote Monitoring PC (RCMSP)
- Stores data in a memory device
- Displays data using graphical interface (GUI)
- Analyses data, compares with normal mode and warns of abnormal deviations.

The Railwise system has a distributed architecture at four levels:
1. Physical layer: sensors and Fieldbus nodes
2. Data Link layer: defines the methods used to transmit and receive digital data, WorldFIP PC card
3. Transport layer: manages the flow control of data, employs WorldFIP protocol
4. Presentation layer: presents graphical information

Railwise employs WorldFIP Fieldbus distributed data acquisition technology. The network consists of track side elements (Fieldbus nodes and sensors, track side PC), server PC (RCMSP), data communication (modems), data presentation and analysis.

**Figure 1 – A Typical Fieldbus Node Architecture**

**Fieldbus nodes** - The transducers are connected to the Fieldbus network using WorldFIP communication protocol (specified in IEC-61158). Two different types of nodes have been designed – the digital node and the analogue node. The digital node acquires digital data from spare relay contacts and digital output signals from external monitoring equipment. Each analogue node is designed to provide an independent isolated power supply to two transducers and transmit the transducers’ output across the Fieldbus network to the PC. The analogue nodes are situated locally to the sensors (<500 m).

**Trackside PC (TSPC):** An on-site PC is used to control the network and process the data acquired. It should be housed in a floor standing lockable cabinet to provide security. The PC is required to transmit and receive data using a telephone modem. Industrial PC uses RAID technology for data backup (see Figure 2 – TSPC).

**Figure 2 – Railwise RCM Track-side PC**

**Server PC (RCMSP):** RCMSP connected to TSPC using ADSL, ISDN or plain old telephone system. It can be used as auto backup, alarm or data controller. The Server PC allows the publication of information on the Internet via an XML interface.

**Data Presentation and Analysis.** RailWISE provides the user with a mimic layout of the selected junction assets with the ability to dial up the site PC to retrieve data. Once the data has been downloaded to the system, the data files can be selected and analysed using CD player style controls to play/rewind events or to single step through the data. There are also options to edit the configuration file, view/print digital lists, export configuration files and view live analogue data.

**Figure 3 – A Typical Railwise GUI Front-end**
RailWISE System Boundary - The RailWISE System consists of appropriate transducers, Fieldbus nodes and network power supply, network cables, TSPC and RCMSP. The transducer signal is converted into appropriate data to be transmitted to the WorldFIP PC card using the WorldFIP interface. A PCA1001 Fieldbus node circuit board is used to convert data. A PCA1000 Fieldbus power supply interface board is designed to convert voltage into a form that can be used to supply the system. RailWISE is a non-intrusive overlay system and will receive data through previously approved transducers and external monitoring systems.

3 SYSTEM HAZID
The purpose of the HAZID workshop was to identify hazards related to the Railwise (RCM) for a typical application on London Underground Limited (LUL) infrastructure. The objective was to ensure that there was sufficient evidence in support of Railwise (RCM) safety argument that system-level hazards are identified, eliminated, mitigated or controlled to an As Low As Reasonably Practicable (ALARP) level.

3.1 Hazard Analysis Process
The hazard identification process as defined in the Yellow Book [1] and EN50126 [2] was used. The step adopted here identifies:
- Each hazard and its cause (causal factor/precursor).
- Hazard consequences (hazardous events).
- Any existing controls over the hazard.
- Scenario that best describes the hazard.
- The impact of the hazard to the overall risk.
- Mitigation/barriers (additional potential controls) over the hazard and its likely consequences.

Harmful consequences of hazard (incidents and accidents, also known as ‘hazardous events’) were grouped as follows: train collisions; train derailment; collision with an object on the line; passenger injury/loss and/or worker injury/loss.

3.2 Hazard Analysis Guidewords
The following guidewords were used to prompt the identification of hazards during the HAZID brainstorming sessions:
A) Lifecycle issues: An activity checklist was used for each lifecycle phase:
- Design (FMECA, EMC Tests)
- Installation (cabling)
- Test & Commissioning
- Operation (Manuals, Strategy, Training)
- Maintenance (Manuals, Strategy, Training)
- Decommissioning and Disposal
B) Interfaces (e.g., with Signalling or other assets): Example factors/guidewords used:
- Sensors – Equipment Being Monitored
- Sensors – Trackside PC

- Trackside PC – Server PC
- Signalling Centre – RTC LAN
- Data Centre - RCMSP
- Alarm Centre – RCMSP
- RCMSP – TSPC Comms

Figure 4 - RCM System External Interfaces

3.3 HAZID Session Stakeholders
The following competencies provided an input into the hazard analysis process:
- Supplier Representatives - Those with experience of system design, development and installation
- Safety Approval experts - Those from the infrastructure owner/operator/maintainer, with experience of the approval process for non-safety-critical RCM equipment
- Safety and Risk Management Consultants - rcm2 limited acting also as facilitators

3.4 Safety Risk Matrix
Each hazard was assessed for its probability of occurrence and severity of its consequences. This allowed the identification of the risk the hazard contributed to the overall risk of the installation and operation of the system. The assessment was carried out by assigning each hazard with a probability and severity rating as detailed in Table 1, which in turn determined the overall risk rating.

Table 1 Probability that the Hazard Will Occur

<table>
<thead>
<tr>
<th>Category</th>
<th>Rating</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent</td>
<td>6</td>
<td>Continually experienced</td>
</tr>
<tr>
<td>Probable</td>
<td>5</td>
<td>Several times, often</td>
</tr>
<tr>
<td>Occasional</td>
<td>4</td>
<td>Several times but not often</td>
</tr>
<tr>
<td>Remote</td>
<td>3</td>
<td>Sometimes during lifecycle</td>
</tr>
<tr>
<td>Improbable</td>
<td>2</td>
<td>Unlikely to occur but possible though exceptional</td>
</tr>
<tr>
<td>Incredible</td>
<td>1</td>
<td>Extremely unlikely or never</td>
</tr>
</tbody>
</table>
It is often common practice for participants at HAZID session to agree a range of values to which each of these probabilities refer to. For example, “incredible” may refer to a potential accident or harmful incident once in 100 years and “improbable” once in a decade. Generally you may draw on analogy between a new hazard, a similar one, and ask the question that in the past say 10 years how many such incidents have occurred and therefore try to guesstimate the probability of the occurrence of the new hazard in the future.

Table 2 Severity - The Level of Impact the Hazard Will Have

<table>
<thead>
<tr>
<th>Severity Level</th>
<th>Rating</th>
<th>Consequence</th>
<th>Consequence to service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catastrophic</td>
<td>4</td>
<td>Fatalities and/or major damage to the environment</td>
<td>Political (nation-wide or industry-wide)</td>
</tr>
<tr>
<td>Critical</td>
<td>3</td>
<td>A fatality and/or severe injury and/or environmental damage</td>
<td>Loss of a major system</td>
</tr>
<tr>
<td>Marginal</td>
<td>2</td>
<td>Minor injury and/or significant threat to the environment</td>
<td>Severe system damage</td>
</tr>
<tr>
<td>Minor</td>
<td>1</td>
<td>Possible minor injury</td>
<td>Minor system damage</td>
</tr>
</tbody>
</table>

The Overall Risk Rating (ORR) is a summation of the probability and severity ratings and determines the risk level of the hazard. The summation is based on the fact that each of the severity and probability ratings represent an order of magnitude increase from its predecessor, hence addition rather than multiplication of values based on logarithmic scales. An ORR of 4 or less is acceptable, in which case no action is required; between 5 and 7 the risk is tolerable; and above 7 it is intolerable. In the last two cases actions have to be taken to try to reduce the risk to be as low as the ALARP.

For Railwise RCM system case study, typical hazards included:
- Electrical shock to the technician from high voltage sources
- Electro-magnetic interference (EMI) issues
- RCM signals interfering with safety-critical signals as a result of bare cables, or RCM faults transferring incorrect potentials to safety-critical signaling circuitry

<table>
<thead>
<tr>
<th>Occurrence</th>
<th>Risk Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent</td>
<td>7 Tolerable</td>
</tr>
<tr>
<td>Probable</td>
<td>6 Tolerable</td>
</tr>
<tr>
<td>Occasional</td>
<td>5 Tolerable</td>
</tr>
<tr>
<td>Remote</td>
<td>4 Acceptable</td>
</tr>
<tr>
<td>Improbable</td>
<td>3 Acceptable</td>
</tr>
<tr>
<td>Incredible</td>
<td>2 Acceptable</td>
</tr>
</tbody>
</table>

Table 3 Risk Assessment

4 SAFETY REQUIREMENTS
A range of generic safety regulations and specific safety requirements apply to any novel electrical, electronic or programmable electronic system (EEPES). It is often a very time consuming exercise for any safety engineer to have to go through all existing regulations and standards in order to identify the ones that are applicable to the project in hand and to ensure that a complete, correct, clear, concise and consistent set of requirements are derived for the EEPES under study. A typical generic standard applicable to RCM case study is IEC61508 [3], and a specific industry standard is RT/ES/11304 [4]. In section 7, we show how ISCaDE was used to facilitate this task as well as to allow potential changes/updates of standards to be traced and their impacts reviewed during the product lifecycle. It must also be noted that during the HAZID, specific new requirements may be derived to eliminate, mitigate or control newly identified hazards. It may also be useful to have a short-hand list of the most relevant safety requirements pertinent to the EEPES under study at the HAZID session in order to prompt the participants not to miss the obvious hazards that the requirements are designed to manage.

5 SAFETY CASE ARGUMENTATION
The maintenance of a hazard log, risk assessment and compliance with safety requirements are now the cornerstones of any system safety management standard for safety-critical industries [e.g., 5]. These, however, are not sufficient for addressing the central claim that the system is safe for its intended mission, and most Standards now require the production of a System Safety Case. The Safety Case shall contain a structured argument demonstrating that the evidence contained therein is sufficient to show that the system is safe. A safety case is often a larger requirement and implies a set of arguments
and evidence to support a central claim and a structured set of (associated) subclaims. A safety case, therefore, consists of

- **Goals/Claims** – An explicit set of objectives (goals/claims) about the system, whether an undertaking, project or product. These are safety requirements that are adequate and shall be met in a given context (application/environment) based on well-defined validation criteria.

- **Evidence** – Supporting processes and documents such as risk modelling, hazard identification, risk reduction measures, quality and safety management system and audit reports

- **Arguments** – A set of arguments that link the evidence to the goals (claims), together with any underlying assumptions and judgements

Various graphical notations have been proposed to support the development and presentation of system safety cases. These notations are readily used at the early safety case planning phase to identify focus areas needing attention in terms of argumentation and evidence gathering. They are alternatively used at the safety audit and approval phases by independent assessors to find their way logically through a myriad of documents pertaining to a particular system safety case. Safety case notations are discussed in Section 6.5 below.

### 6. INTEGRATED DATABASE ENVIRONMENT

ISCaDE is a networked software environment that uses the DOORS (Dynamic Object-Oriented Requirement System) database as its platform. It combines the features of a multi-user, multi-access, object-oriented database and graphical presentation capabilities in an integrated environment that marries different safety case development techniques: GSN with Hazard Log and traceability to safety requirements/standards compliance.

#### 6.1 Safety Standards & Safety Requirements Capture

A common challenge to the development of a product safety case is to identify the applicable safety standards and legislation and to develop new safety requirements that should drive the design and other phases of the product development. This may be an iterative process and needs to be managed throughout the product’s lifecycle. ISCaDE provides an ‘Initialise Standard’ functionality and allows the whole standard to be imported but only relevant clauses marked as safety requirements and saved as a view for future traceability purposes. A structured Word document is therefore turned into a database table and additional attributes such as validation and verification criteria and tests may be assigned to each safety requirement (Fig. 5).

#### 6.2 The Hazard Log and Hazard Log Form

ISCaDE allows a configurable hazard log with an easy to use data entry form. Each hazard is given a unique identifier and as a DOORS ‘Object’ has properties that include cause, scenarios leading to consequences (accidents and incidents), probability and severity of accidents and the overall risk rating, mitigation and controls, the state of each hazard, actions recorded and the actionees (Fig. 6).

#### 6.3 Hazards/Safety Requirements Gap Analysis

The ability to provide many to many traceability links between hazards and safety requirements is an advantageous feature of the ISCaDE environment. This allows identification of gaps, on the one hand development of new requirements that control/mitigate hazards and on the other ensuring that all hazards that existing standards imply are identified and managed.

#### 6.4 Safety Risk classification Matrix

It is often important for engineers to monitor progress with safety and hazard management at the system level in order to present it to management. ISCaDE produces a system safety risk classification matrix automatically from the snap-shot information stored in the system.
hazard log (similar to Table 3).

6.5 Safety Case Notations

ISCaDE allows automatic production of safety case diagrams from a safety case notation structures such as the following:

a) **Goal-Structuring Notation** (GSN) is a graphical approach to presenting the structure of a safety argument. Goal hierarchies consist of: Goals – a requirement, target or constraint to be met by the system; Strategies – rules to be invoked in the solution of goals; Context; Models; Justification; Assumption and Solutions – evidence, analysis, design review and audit report.

b) **Adelard Safety Case Development Manual** (ASCAD) is a total safety case development strategy. It is based on Evidence-Argument-Claim structure: Claims about the properties of the system; Evidence used as basis for the safety argument; and Argument that links the evidence to the claims via a series of inference rules.

c) **Weighted Factor Analysis** (WeFA) is a graphical presentation of drivers and inhibitors to a top safety goal or objective. Each driver/inhibitor is in turn a subgoal with a different positive/negative contribution to the higher level goal, represented by a weighting factor.

Each of the above notations has its particular strengths. The WeFA model allows one to view the opportunities for improving safety the new product entails as well as the threats (managing hazards). ASCAD uses the terminology (evidence, argument) readily used by engineers in their safety case documentation. GSN, on the other hand, benefits from the ability to assign attributes such as Context, Justification and Assumptions to goals (claims) and strategy (arguments) and the ‘solution’ is where supportive ‘evidence’ of compliance or argumentation is phrased. Figure 7 shows an ISCaDE-generated GSN diagram for the Railwise system.

Figure 7 – ISCaDE GSN for Railwise

7. CONCLUSIONS

The overall approach advocated in this paper is to view the safety approval process as synergetic to, and in parallel with, the systems engineering and requirements management processes. In other words, the safety paradigm is an intrinsic element of the overarching problem: is the system ‘fit for purpose’? And is the safety managed accordingly, i.e., do the safety requirements (that control/mitigate hazard), design features, implementation etc. meet the system mission? This approach was demonstrated for the RCM case study using Integrated Safety Case Development Environment (ISCaDE), a commercial off-the-shelf (COTS) software package. ISCaDE extends the features of the DOORS requirements management database to cover the techniques and processes widely used for safety management. The benefits and challenges of using such an integrated environment may be summarised as follows:

- No duplication or missing effort as all project information including safety is managed within a single object-oriented database.
- Transparency of information to all team members within a secure multi-user multi-access environment.
- Graphical presentation of safety information automatically from the structured data for the benefit of all stakeholders including managers to monitor progress with technical aspects of the product/project.
- Automatic traceability and a full audit trail between all elements of a system safety case and the processes and techniques used in its development and maintenance.
- Encouraging and facilitating cooperative teamwork.
- Saving considerable time and money, ultimately leading to a safer and less expensive end-product.

8. REFERENCES


Biography

Dr Saeed Fararooy is the managing director of rcm2 limited, a small high-tech UK company offering system safety and reliability consultancy, training and systems/IT engineering/integration services to transportation and other safety-critical industries. He is an electronic control
systems engineer with over 22 years experience in a range of industries from manufacturing and process to transportation.
A Taxonomy of Safety-Related Requirements

Donald Firesmith
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213 USA
+1 (412) 268-6874
dgf@sei.cmu.edu

ABSTRACT
As software-intensive systems become more pervasive, more and more safety-critical systems are being developed and deployed. Yet when most people think about safety requirements, they think of safety-critical functional requirements, which are requirements that have critical safety ramifications if not correctly implemented. However, there are actually four major classifications of safety-related requirements: (1) pure safety requirements, (2) safety-significant requirements, (3) safety constraints, and (4) requirements for safety systems. This paper describes a taxonomy of these different kinds of safety-related requirements, and clearly and briefly defines and describes each of the above categories of safety-related requirements.

Keywords
Safety, safety-critical, safety requirement, safety constraint, safety categorization, safety engineering, requirements engineering

1 SAFETY-RELATED REQUIREMENTS
As software-intensive systems become more pervasive, more and more safety-critical systems are being developed and deployed. Yet when most requirements engineers think about safety requirements, they think of safety-critical functional requirements, which are functional requirements that have critical safety ramifications. Such safety-critical functional requirements typically can cause serious accidents if incorrectly implemented, although it is also possible to unintentionally specify requirements that can cause accidents even if they are properly implemented. Many people also incorrectly confuse safety requirements with reliability requirements, even though it is possible to have a safe system that is unreliable (e.g., it never does anything) and an unreliable system that is safe (e.g., its failures do not cause any harm).

A major theme of this paper is that there exists a large taxonomy of safety-related requirements, only a few of which are typically identified, analyzed, and specified on most projects. Most safety and requirements engineers will be familiar with the term “safety-critical requirements”, and some may well be familiar with safety requirements limiting the existence of specific hazards. Unfortunately, there are many other kinds of safety-related requirements that should be considered when developing systems with significant safety implications.

In the taxonomy presented in this paper, there are actually four major classifications of safety-related requirements:

1. Safety requirements – any requirements that specify mandatory amounts of a subfactor of the safety quality factor (i.e., requirements to protect assets from accidental harm, detect safety incidents, and respond to safety incidents).

2. Safety-significant requirements – any non-safety primary mission requirement that can cause hazards and safety incidents. They include functional requirements, data requirements, interface requirements, and non-safety quality requirements (e.g., accuracy, capacity, performance, reliability, robustness, or security) that are necessary to meet the primary mission of the system.

3. Safety constraints – any constraints (i.e., engineering decisions that have been selected to be mandated as requirements) that directly impact safety.

4. Safety system requirements – any requirements for safety systems as major components of systems of systems.

The primary purpose of safety engineering is to protect valuable assets from accidental harm, which is harm that

1 It is very difficult to find intuitively obvious names that clearly distinguish these four types of safety-related requirements, especially the first two. There is little in the way of standard naming conventions because the different types are often confused with each other or themselves misleading (e.g., all safety-significant requirements are not safety-critical).

2 This differentiates safety from security, which is concerned with protecting valuable assets from malicious and unauthorized harm. Also, security must protect valuable assets from ever-present threats, safety deals with
is unplanned and unintended but not necessarily unexpected. Many accidents are caused by problems with system and software requirements, and “empirical evidence seems to validate the commonly stated hypothesis that the majority of safety problems arise from software requirements and not coding errors” [2]. Accidents typically arise from the occurrence of rare hazards, which are combinations of conditions that increase the likelihood of accidents causing harm to valuable assets such as people, property, or the environment. Requirements specifications are typically incomplete because they do not specify how the system should avoid or eliminate hazards and how the system should behave when hazards or safety incidents occur. Requirements specifications are also typically incomplete in that they usually do not define how the system should behave in all reasonable combinations of states or how the system should handle exceptional circumstances, such as mandatory exception handling.

The safety team of safety analysts and engineers typically produce a safety program plan and perform various types of safety analyses using such techniques as fault trees analysis (FTA), event tree analysis(ETA), hazard cause and effects analysis (HCEA), and failure mode and effect analysis (FMEA). However, their efforts are usually not integrated into the requirements specifications, and this makes it difficult to ensure that the architecture incorporates the appropriate safeguards.

2 SAFETY REQUIREMENTS

Safety requirements are those safety-related requirements that are specifically engineered to achieve mandatory amounts of a subfactor of the safety quality factor (i.e., requirements to protect assets from accidental harm, detect safety incidents, and respond to safety incidents). Whereas normal requirements specify what the system shall do or make happen, safety requirements specify what the system shall not do or prevent from happening [3].

Safety requirements are those requirements specifically engineered to achieve a specific minimum level of the quality attribute “safety”. Whereas normal functional, data, and interface requirements specify what the system shall do or make happen, safety requirements specify what the system shall not do or prevent from happening [3].

Many safety requirements are specified to directly protect valuable assets from harm due to accidents resulting from hazards that cause safety risks. Whereas a gram of relatively rare hazards, so that security attacks are common whereas safety incidents (i.e., accidents and near misses) should be very infrequent [1]. Thus, security requirements must protect against common dangers whereas safety requirements protect against hopefully very rare dangers, a contrast that has major implications concerning the verifiability of safety requirements.

prevention is worth a kilogram of cure, sometimes accidental harm cannot be prevented and we must resort to cure. Thus, there should typically also be safety requirements for detecting the occurrence of safety incidents (an accident or near miss) and properly reacting to the occurrence of safety incidents:

- **Protect valuable assets requirements:**
  - **Asset/harm requirements.** This type of safety requirement sets a maximum acceptable limit on the amount of harm that may occur to an asset. The specified harm may be restricted to a specific asset or a type of asset. It can also be limited to a specific level (severity) of harm. For example, “On average, the automated airport transport system shall not accidentally injure more than one passenger seriously enough to require hospitalization per 50,000 passenger trips.”
  - **Accident requirements.** This type of safety requirement sets a maximum acceptable limit on the number or frequency of safety incidents (especially accidents) or types of safety incidents. For example, “On average, the automated airport transport system shall not allow more than one passenger to fall out of an open subway door per 50,000 passenger trips.”
  - **Hazard requirements.** This type of safety requirement sets a maximum acceptable limit on the frequency or duration of hazards (e.g., by type or specific hazard). For example, when specifying requirements for an automated subway system connecting the terminals of an airport, the combination of a moving train with open doors (two conditions) is clearly a hazard where the potential harm is to both passengers and their luggage (two types of valuable assets). A ‘protect valuable assets from hazard’ safety requirement could be: “On average, the automated airport transit system shall not allow a subway train to be moving when one or more of its doors are open (the quality criterion) for a duration of more than one second more than once per year (the quality measure).” [5]
  - **Safety risk requirements.** This type of safety requirement sets a maximum acceptable limit on the level of risks (e.g., by risk category or by specific risk). For example, “The automated airport transit system shall not have any safety risks estimated as intolerable (i.e., safety integrity level = 5).”

- **Detect safety incident requirements:**
  This type of safety requirement specifies how a system must detect safety incidents that occur in spite of the preceding requirements for protecting valuable assets. For example, “On average, the automated airport transport system shall not accidentally injure more than
one passenger seriously enough to require hospitalization per 50,000 passenger trips.”

- **React to safety incident requirements:**
  Safety requirements are often relatively reusable, especially within an application domain and across members of a product line. As quality requirements, safety requirements are typically of the form of a system-specific quality criterion together with a minimum or maximum required amount of an associated quality measure [4]. This structure means that safety requirements can often be written as instances of parameterized generic safety requirement templates. For example:

<table>
<thead>
<tr>
<th>Type of Safety Requirement</th>
<th>Form of Parameterized Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prevention of Accidental Harm to Valuable Asset</td>
<td>The system shall not [cause</td>
</tr>
<tr>
<td>Prevention of Safety Incidents (esp. Accidents)</td>
<td>The system shall not [cause</td>
</tr>
<tr>
<td>Prevention of Hazards</td>
<td>The system shall not [cause</td>
</tr>
<tr>
<td>Prevention of Safety Risk</td>
<td>The system shall not [cause</td>
</tr>
<tr>
<td>Detection of Violation of Prevention</td>
<td>The system shall detect [accidental harm</td>
</tr>
<tr>
<td>Reaction to Violation of Prevention</td>
<td>When the system detects [accidental harm</td>
</tr>
</tbody>
</table>

### 3 SAFETY-SIGNIFICANT REQUIREMENTS

A safety-significant requirement is any non-safety primary mission requirement that can cause hazards and safety incidents (i.e., an accident or near miss). Safety-significant requirements include functional requirements, data requirements, interface requirements, and non-safety quality requirements that are necessary to meet the primary mission of the system. Thus, safety-significant requirements have associated safety risks.

This paper recommends a relatively standard approach to categorizing safety risks in terms of accident severities and likelihoods. The resulting categorizations lay a foundation for the taxonomy of the different kinds of safety-significant requirements.

At the beginning of a project before asset, hazard, and safety risk analysis are performed, the safety team typically categorizes accident/hazard severities, accident/hazard likelihoods, and associated safety risks. These categorizations are then used during the rest of safety engineering as well as during the engineering of various types of safety-significant requirements.

#### Accident Severity Categorization

The severity of the harm that an accident can cause to a valuable asset varies from inconsequential to catastrophic. To make this continuum of damage manageable, the safety team typically categorizes it into a small number of severity levels. Accident severity levels are typically based on the worst credible impact of a type of accident or of an accident resulting from a given hazard. Instead of emphasizing only health safety, the safety engineers should take care to also include harm to each kind of valuable asset (i.e., life, property, and the environment). The actual levels and the boundaries between levels naturally will vary from project to project, although projects within certain application domains may share accident severity levels specified by international, military, or industry standards [6], [7], [8], [9], and [10]. An example of such an accident/hazard severity categorization is found in Table 1.

#### Accident/Hazard Likelihood Categorization

The probability of accidents can vary from relatively high to essentially zero. However, accident and hazard probabilities are often very difficult if not impossible to accurately and precisely estimate during the development of a complex system. This is especially true if the system contains significant amounts of software because the failure modes are discontinuous and thus difficult to predict. Therefore, accident and hazard likelihoods are often divided into a small number of categories, typically having intuitive if somewhat ambiguous definitions. Cautious safety engineers take care to ensure that their likelihood categories are based on a relevant taxonomy of the environment. The actual levels and the boundaries between levels naturally will vary from project to project, although projects within certain application domains may share accident severity levels specified by international, military, or industry standards [6], [7], [8], [9], and [10]. An example of such an accident/hazard severity categorization is found in Table 1.

3 Note that a lack of adequate accuracy, capacity, performance, reliability, robustness, and security can all negatively impact the safety of a system

4 The term “likelihood” is used instead of probability because it better implies the lack of accuracy and precision.
system. Similarly, accident and hazard likelihoods for individual systems should be differentiated from the much higher likelihoods of collections of similar systems. For example, the accident likelihood for an individual aircraft is less than the combined accident likelihood for all aircraft within a fleet. An example of such an accident/hazard likelihood categorization is found in Table 2.

**Safety Risk Categorization**

Using accident severity categories and accident frequency categories, the safety engineers typically produce a matrix of safety risk indices (individual cells in the matrix) that can be grouped into a smaller more manageable number of safety risk categories, a.k.a. safety integrity levels (SILs). An example of such a safety risk matrix is documented in Table 3, whereby its associated safety risk categories are defined in Table 4. Note that the actual values of the safety risk indices and the symmetry of the table will vary between projects, application domains, and associated international, military, and application domain specific standards.

<table>
<thead>
<tr>
<th>Severity Level</th>
<th>Definition in terms of the Level of Harm to Valuable Assets</th>
</tr>
</thead>
</table>
| **Catastrophic** | • Loss of life (e.g., members of the public, users, and operators)  
• Life threatening or permanently disabling injury  
• Loss of system  
• Property/financial loss exceeding $1,000,000  
• Irreversible severe environmental damage that violates a law or regulation |
| **Severe** | • Permanent partial disability  
• Injury or occupational illness requiring hospitalization of at least 3 individuals  
• Loss of subsystem  
• Property/financial loss exceeding $200,000  
• Reversible environmental damage that violates a law or regulation |
| **Major** | • Injury or occupational illness resulting in one or more lost work days  
• Loss of component  
• Property/financial loss exceeding $10,000  
• Mitigable environmental damage where restoration can be accomplished without violating a law or regulation |
| **Minor** | • Injury not resulting in the loss of a work day  
• Property/financial loss less than or equal to $10,000  
• Minimal environmental damage where restoration can be accomplished without violating a law or regulation |
| **None** | No harm is caused to any valuable asset |

Table 1. Example Accident Severity Categories

<table>
<thead>
<tr>
<th>Likelihood Category</th>
<th>Definition of Likelihood Category (for both accidents and hazards for both individual systems and for sets of systems)</th>
</tr>
</thead>
</table>
| **High**            | • An accident will frequently occur during an individual system’s operational lifespan.  
• Rough estimate that an accident will occur during an individual system’s operational lifespan: $10^4 < $1,000,000  
• Loss of system  
• Property/financial loss exceeding $1,000,000  
• Irreversible severe environmental damage that violates a law or regulation |
| **Moderate**         | • Permanent partial disability  
• Injury or occupational illness requiring hospitalization of at least 3 individuals  
• Loss of subsystem  
• Property/financial loss exceeding $200,000  
• Reversible environmental damage that violates a law or regulation |
### Accident/Hazard Likelihood

<table>
<thead>
<tr>
<th>Safety Risk</th>
<th>Accident Severity</th>
<th>Low</th>
<th>Remote</th>
<th>Negligible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catastrophic</td>
<td>Intolerable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Severe</td>
<td>Intolerable</td>
<td>Critical</td>
<td>Major</td>
<td></td>
</tr>
<tr>
<td>Major</td>
<td>Critical</td>
<td>Critical</td>
<td>Major</td>
<td>Moderate</td>
</tr>
<tr>
<td>Minor</td>
<td>Critical</td>
<td>Major</td>
<td>Moderate</td>
<td>Minor</td>
</tr>
<tr>
<td>None</td>
<td>None</td>
<td>None</td>
<td></td>
<td>None</td>
</tr>
</tbody>
</table>

Table 3. Example Safety Risk Matrix

### Requirements

The preceding categorizations lay the foundation for identifying, analyzing, and specifying the safety-significant requirements.

For example, suppose that you are specifying the requirements for an automated subway system connecting the terminals of an airport. Clearly, there will be functional requirements for opening and closing the subway doors and more functional requirements for starting, moving, and stopping at the next terminal. Although it is perfectly okay and even necessary to have the subway doors open or to have the subway move between terminals, it is clearly a hazard to have the doors open while the subway is moving because passengers and their luggage can fall out. Therefore the requirements for opening and closing doors as well as the requirements for starting, moving, and stopping are safety-significant functional requirements.

One common oversimplification is that safety and requirements engineers may consider only functional requirements, forgetting that data requirements, interface requirements, and certain quality requirements (e.g., reliability) can also result in safety hazards and associated safety incidents if not properly implemented. For example, incorrect implementation of data requirements regarding to the maximum permitted acceleration can cause people to fall. Similarly, incorrect implementation of interface requirements for communicating with the security system might let the subway stop at a station that has been closed do to fire or terrorist attack. Finally, poor reliability resulting in a critical failure or poor security resulting in a successful security breach can both lead to accidents.

Another common oversimplification is that safety-significant requirements are all grouped together and referred to as safety-critical requirements. Safety-significant requirements can and should be further classified in terms of the amount of their associated safety risk, which is typically defined as the probability that a hazard exists multiplied by the conditional probability that the hazardous conditions will result in an accident multiplied by maximum harm that the accident can cause. To deal with the resulting continuum of safety risks, they are usually further categorized into a small number of safety integrity levels (SIL) as documented in Table 4. For example, SIL = 5 might mean an intolerably high risk such as a significant probability of catastrophic harm, whereas SIL = 0 might mean that essentially no safety risk exists.

Thus, some functional, data, and interface requirements may have an associated safety risk that is intolerable (e.g., SIL = 5); such requirements should be rejected and thus should not be included as requirements in a requirements specification. Other requirements having a high SIL value (e.g., SIL = 4) are safety-critical, and a safety-critical system is one for which there exists at least one risk with a...
SIL greater or equal to 4 [8]. Because of this level of risk, safety critical systems require a very high safety evidence assurance level (SEAL) for their correct implementation. For example, safety-critical requirements may need to be formally specified using a mathematically precise specification language to enable automatic source code generation of provably safe software. Similarly, semi-formally specification using modeling languages such as state charts and decision trees may be adequate for safety-major (SIL = 3) requirements, whereas non-formal specification using structured textual requirements may be adequate for safety-moderate (SIL = 2) and safety-minor (SIL = 1) requirements. Finally, a SIL value of 0 means that there is essentially no associated safety risk, so that such functional, data, and interface requirements can be considered to be safety-independent requirements.

The identification of safety-significant requirements will depend on (1) the prior existence of functional, data, interface, and quality requirements to be categorized by SIL value, (2) the existence of a list of categorized hazards and associated safety risks, and (3) the performance of an associated safety risk analysis (e.g., based on fault-tree, event-tree, and similar techniques) to identify the safety risks associated with the individual functional, data, and interface requirements.

4 SAFETY CONSTRAINTS
A constraint is a business rule or engineering decision that is treated during requirements engineering as if it were a requirement even though it would ordinarily be made during architecture development, design, implementation, integration, or testing. A safety constraint is often a mandated safety policy or a mandated safeguard such as a hardware interlock, barrier around moving parts, handling procedures for toxic materials, and safety procedures. Whereas safety constraints are in many ways no different than other types of constraints, because they are specified for safety reasons, they are subject to safety certification and accreditation like other safety-related requirements. Some safety constraints are required by a relevant regulation, standard, or law. In fact, some safety constraints merely mandate compliance with such a regulation, standard, or law, and therefore act as a way to group the numerous constraints included in the regulation, standard, or law. On the previously mentioned example airport transit system, the following standards might be mandated: ASCE 21-96 [5] or IEEE P1474.1 [11]. Relevant example safety constraints include:

- “The airport transit system shall comply with Automated People Mover Standards, Part 1, ASCE 21-96.”
- “System safety shall not depend on the correctness of actions taken or procedures used by operating personnel.”
- “No credible single point hardware failure, whether self-revealing or non-self-revealing, shall cause an unsafe condition.”
- “The doors shall be disabled from opening unless: (1) the train is at a designated stopping point within designated tolerances, (2) zero movement is detected, and (3) the train is constrained against movement.”

5 SAFETY SYSTEM REQUIREMENTS
The three types of safety-related requirements that have been discussed so far (i.e., safety-significant requirements, safety requirements, and safety constraints) typically specify aspects of the primary system to be built. However if there is a significant risk associated with the primary system, then it may well have major mandatory safety subsystems or it may be developed in conjunction with one or more safety systems, whereby these safety systems or subsystem only exist to ensure the safety of the primary system. The classic example of such safety systems is the emergency coolant system of a nuclear power plant. Every functional, data, interface, and quality requirement of the safety system may well have significant safety implications. These requirements for the safety system or subsystem are safety system requirements, the fourth and final category of safety-related requirements in our taxonomy.

6 SAFETY REQUIREMENTS TAXONOMY
As illustrated in Figure 1, the preceding four basic classifications of safety-related requirements can be placed into a larger taxonomy of requirements types. At the lowest level of abstraction near the bottom of the figure, all of the system requirements are classified as functional requirements, data requirements, interface requirements, quality requirements, or constraints. In a separate orthogonal inheritance subtree extending to the right, all system requirements (i.e., all functional, data, interface, and quality requirements as well as constraints) can also be classified into main mission system requirements and safety system requirements. Thus, the taxonomy uses multiple classification to produce two independent overlapping inheritance structures.

On the left side of Figure 1, we see that safety-significant requirements are all system functional, data, interface requirements that have safety ramifications combined with all non-safety quality requirements (i.e., all other quality requirements not specifically mandating minimal acceptable amounts of safety such as availability, capacity, performance, portability, security, and usability requirements). The figure also clearly shows that all safety-significant requirements are not safety critical requirements, but only those with safety integrity level 4 (or those with the equivalent program-specific safety integrity level). The top branches of the safety requirements taxonomy classify the safety requirements (those directly specifying an amount of the safety quality factor) into two independent inheritance trees: one tree
addressing the goals of protection, detection, and reaction while the second tree captures the requirements resulting from the first four types of safety analysis. Finally, safety constraints are a subset of constraints that mandate specific safeguards and safety mechanisms.

7 CONCLUSION
There are several different kinds of safety-related requirements and using a standard taxonomy to organize them can have the following benefits:

- The taxonomy can help requirements and safety engineers ensure that no significant types of safety-related requirements fall through the cracks during requirements and safety engineering.
- The taxonomy captures different types of safety-related requirements that are currently being engineered on real projects.
- The different types of safety-related requirements in the taxonomy have different sources (e.g., different stakeholders and documents).
- The different types of safety-related requirements are elicited and analyzed differently. Thus, “limit harm” safety requirements can be derived from business goals. “Limit accidents/hazards/risks” safety requirements can be derived from an asset/hazard/risk analysis. Safety significant requirements are typically derived the same way and at the same time as any other functional, data, and interface requirements; it is only after they exist that they are analyzed for their safety implications. Safety constraints typically come from regulations, laws, standards, and best industry practices.
- The different types of safety-related requirements are specified differently. Safety-significant requirements are primarily specified with the other functional requirements (and data and interface requirements). Their safety aspects are specified as metadata (attributes). On the other hand, safety requirements are typically specified with the other quality requirements. Similarly, safety constraints are typically specified with the other constraints. Finally, safety system requirements are specified separately in the requirements specification for the safety system or subsystem.
- The different types of safety-related requirements typically have different reuse potentials. Safety requirements and safety constraints are often very reusable. For example, safety requirements can be reused as parameterized templates based on the quality model used [12]. Safety-significant requirements tend to be much more application specific and less reusable.

FUTURE WORK
This taxonomy is being used as the foundation for an Independent Research and Development (IRAD) project at the SEI which is producing technical notes on reusable safety requirements and a process for engineering safety requirements. Other future work to be considered might include researching the following questions:

- How do the different kinds of safety-related requirement vary in terms of their reuse potential?
- How do the different kinds of safety-related requirements vary in terms of their optimum identification, analysis, and specification techniques?
- How should the efforts of the safety and requirements teams be better coordinated?

ACKNOWLEDGEMENTS
I would like to thank my colleagues Peter Capell, Tim Morrow, and Mary Popeck of the SEI as well as Daniel Berry and Paul Tiplady for their useful review comments. Of course, any remaining defects are my own.

STIPULATIONS
The views and conclusions contained in this position paper are solely those of the author and should not be interpreted as representing official policies, either expressed or implied, of the Software Engineering Institute, Carnegie Mellon University, the U.S. Air Force, the U.S. Department of Defense, or the U.S. Government.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

REFERENCES


Using Abuse Frames to Bound the Scope of Security Problems

Luncheng Lin  Bashar Nuseibeh  Darrel Ince

Department of Computing  
The Open University  
Walton Hall  
Milton Keynes MK7 6AA, U.K.  
E-mail: {L.C.Lin, B.Nuseibeh, D.C.Ince}@open.ac.uk

ABSTRACT

Security problems arise from the concern for protecting assets from threats. In a systems development process, the security protection of a system is described by security requirements, identified from the analysis of the threats to the system. However, as it is often not possible to obtain a full system description until late in the Requirements Engineering (RE) process, a security problem often can to be described incrementally in the context of a bounded scope, that is, one containing only the domains relevant to a particular functionality of the full system. By binding the scope of a security problem, it can be described more explicitly and precisely, thereby facilitating the identification and analysis of threats and vulnerabilities, which in turn drives the elicitation and elaboration of security requirements. In this paper, we illustrate our approach based on abuse frames and suggest how it can provide a means for structuring security problems to facilitate an incremental development process.

Keywords

Security requirements engineering, threat analysis, abuse frames, problem frames, twin-peak process model

1. INTRODUCTION

In Mandarin, the terms safety and security are denoted by the same characters: 安全 [an-quan], which mean “the protection or preservation of the well being of a person or system”. While a distinction can be made in English, in systems engineering, safety and security concerns are often closely related. Security engineering is mostly concerned with the protection of systems from potential abuse by malicious users, whereas safety engineering is concerned with protection from undesirable phenomena caused by accidental events. Of course, security problems can also arise accidentally and can lead to safety concerns, so contributions to security requirements engineering may well improve safety.

This paper focuses on developing an RE-based threat analysis technique. In systems requirements engineering, a security problem can be addressed by identifying and analysing the threats to a system, and specifying appropriate security requirements to counter those threats. Traditional threat analysis techniques have focused on system design [7]. Threats are identified by using arbitrary threat lists, or by examining system components in tabular forms [5] or in hierarchical structures [2]. Security attacks are usually derived through brainstorming (e.g., [3]), or more systematically, by attack trees [20]. These techniques are often applied when detailed descriptions of the full system design are available.

In RE, misuse cases [1, 22], abuse cases [17], and reusable patterns [21] extend the UML notation for modelling misuses of a system. Software fault trees [9] from the hazard analysis literature [14] have also been applied for deriving security attacks.

van Lamsweerde [12] suggests modelling the notion of malicious intent as an anti-goal in the KAOS framework. An anti-goal is a description of the undesirable phenomena that is to be prevented from happening. An anti-goal model is derived from security goals using obstacle analysis [13] and is elaborated by regressing through domain knowledge. Vulnerabilities are identified as the terminal goals.

Using the $i^*$ framework, Liu et al. [16] take an organisational view by modelling actor dependency relationships as softgoals to be satisfied. Malicious users are modelled as malicious agents, and their attacks are modelled as negative contributions that obstruct these softgoals. Vulnerabilities are identified as the ability of a malicious agent to break a dependency relationship.

For most of these techniques, the identification of vulnerabilities often depends on the knowledge available about the domain properties and of the system context of the envisioned system. As it is often not possible to obtain a full system description until late in the requirements process, an abstract view of a system context should be obtained by identifying and describing domains and their interactions necessary in achieving some part of the system functionality. The earlier such description is available, the earlier threat analysis can be performed, thereby facilitating a better integration of threat analysis and system’s requirements elaboration. This approach is similar to that of adopted Cheheyel et al. [6] but focuses on the requirements process rather than design. We illustrate our abuse frames technique and suggest how it can provide a means for structuring the scope of security problems to facilitate early threat analysis in the requirements process.

2. ABUSE FRAMES AND TERMINOLOGY

Abuse frames consider threats to a system from the viewpoint of a malicious user. We define a threat to be the potential for use of domains in the system to cause harm.
An attack is defined as a realisation of a threat. In [8], we introduced the notion of anti-requirements (AR) to represent the requirements of users with malicious intent, that is, an anti-requirement specifies the undesirable phenomena in the system that must be prevented from happening. An anti-requirement differs from a normal system’s requirement in that an anti-requirement only needs to be existentially satisfied.

We incorporate anti-requirements into abuse frames to represent a security threat [15]. Abuse frames elaborate on the principles of problem frame decomposition and adopt the problem frames notation [10]. The notation is an extended context diagrams notation for structuring a system development problem as a set of subproblems with each subproblem represented as a problem frame diagram. A problem frame diagram relates a projection of the system to a requirement for achieving some part of the system functionality. Through the problem frame notation, each subproblem can be analysed individually. The reader is referred to [10] for detailed explanation and worked examples of the problem frame notation.

Figure 1 is one generic structure of an abuse frame. The system context represents a projection of the system that can be the whole or a part of the system which is under attack. The plain rectangles represent problem domains.

Figure 1: A generic abuse frame diagram.

The Machine domain, which is represented by the rectangle with two vertical stripes, acts as the interface between the Malicious User and the Asset. The Malicious User domain represents the domain that is imposing the threat. Phenomena shared between two domains are represented by an annotated line connecting the two domains.

The anti-requirement, AR, indicated by the dashed oval, specifies the observable and undesirable phenomena E1 in the Asset domain as the result of E2. The interface of the Machine describes the relation between the E3 of the Malicious User and the E2 of the Asset that, in conjunction with the properties of the Malicious User and Asset domains, will satisfy the anti-requirement. In cases where the threat is realised without the active participation of a malicious user, the malicious user domain may be omitted in the diagram.

3. THREAT ANALYSIS USING ABUSE FRAMES

In order to facilitate the integration of system’s requirements elaboration and threat analysis, we suggest a twin-peaks style [18] process model (Figure 2) in which the partial systems specified by the systems requirements are repeatedly evaluated through threat analysis to support security requirements elaboration and elicitation.

![Figure 2: An incremental process of systems requirements elaboration and abuse frames analysis](image)

Initially, security objectives (e.g., confidentiality, integrity, availability and authenticity) are derived by identifying any critical assets that need to be protected in the envisioned system. The initial requirement statement will consist of a description of the functionality of the system, together with a description of these security objectives. Our threat analysis technique essentially comprises five steps:

**Scoping the problem and identify the subproblems:** from the initial problem statement, identify the subproblems and map out their problem frames diagrams. Describe the security concerns as security constraints on the functionality to be achieved in each problem frame diagram.

**Identifying the threats and constructing abuse frames:** anti-requirements can be identified as the phenomena that would result from failing to address the identified security concerns. Initially they can be obtained by negating the security constraints on the each problem frame. The identified anti-requirement is then captured in an abuse frame to represent the threat to the system context bounded by the problem frame diagram.

**Identifying security vulnerabilities:** from the abuse frame diagram, identify the security vulnerabilities as the properties in terms of the behaviours of the domains for which the anti-requirement can be existentially satisfied.

**Addressing security vulnerabilities:** Security vulnerabilities identified will need to be addressed appropriately. Any design decisions made should be checked for preserving the security properties as prescribed by the security goals/requirements. As focus of this paper is on the structuring of security...
problems, this last step will not be discussed in detail in this paper.

It is important for the software developer to verify whether a newly incorporated security requirement or a modification of problem descriptions has potentially introduced new vulnerabilities into the system. This process will involve incorporating the changes into the existing problem frame diagram, and then iterate the abuse frame analysis.

4. A BANKING SYSTEM EXAMPLE

Consider the example problem of developing an online banking system in [12]. The primary function of the online banking system is to allow customers to manage their account online. The original problem can be decomposed into three subproblems: the Customer Identifier, the Account Viewer, and the Transaction Manager. Due to page limitation, only the analysis of the customer identifier subproblem is discussed.

The customer identifier subproblem (Figure 3) is concerned with authenticating and authorising the customer of the online banking system.

![Figure 3: A problem frame diagram for the customer identified subproblem.](image)

The **Auth** domain contains customer IDs and the associated passwords. The **Result** domain reflects whether the current customer has been authenticated and authorised. It acts as a temporary place holder for the status of the customer’s authorisation which is to be referred to by the other two subproblems.

Having defined the context of the subproblem in Figure 3, the next step is to identify and represent the threat to it. A correctly functioning Customer Identifier machine must prevent from giving authorisation to unauthorised users. We call the anti-requirement that specifies this undesirable phenomena *false customer authorisation*.

To construct the abuse frame capturing this anti-requirement, we first investigate how a malicious user (hereafter indicated as **MU** in abuse frame diagrams) interacts with a system in the event of an attack. A malicious user can initiate an *internal* or *external* attack on the system. In an internal attack, a malicious user interacts with the system through the phenomena as described in the original problem diagram. Figure 4 is the abuse frame that represents the potential misuse of the Customer Identifier by an internal malicious user. The reader will also note that the anti-requirement also refers to the **Auth** domain for the definition of an unauthorised user.

![Figure 4: An abuse frames diagram representing the threat imposed by an internal malicious user.](image)

On the other hand, in an external attack a malicious user interacts with the system through unexpected or unintended shared phenomena, i.e. those outside the scope of the original problem frame diagram. To achieve this, the malicious user may require an external domain that exploits the vulnerabilities arising from the domains properties that have not been considered during our initial problem analysis. This is shown as the abuse frame diagram on the right in Figure 4. The large box indicates the context of the existing system. The **External Domain**, controlled by the malicious user through **e**, shares external interactions **f** with the existing system. As these are not included in our original problem statement, the **External Domain**, and the phenomena **f** and **e** have to be identified during the course of abuse frame realisation.

The threat represented by our abuse frames is realised when its anti-requirement can be existentially satisfied. As exemplified in Figures 4 and 5, abuse frames define the scope and the span of the system context by providing an abstract view of the domains and their interactions using the context diagram notation. Existing forward- and backward-tracing techniques such as threat trees [19] and variants of HAZOP [11] can be applied to identify a chain of domain interactions that result in the satisfaction of the anti-requirement. The vulnerability is then identified...
as the property of the system (excluding the MU) that allows the sequence of interactions to take place. The details of the analysis are omitted here for brevity and a summary of the results are presented in the following table.

Table 1: Threat realisation, exploited vulnerabilities and attack examples for the left frame diagram in Figure 4.

<table>
<thead>
<tr>
<th>Realisation of the threat</th>
<th>Example</th>
<th>Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password cracking</td>
<td>Brute force search.</td>
<td>Unlimited tries of password entries</td>
</tr>
<tr>
<td></td>
<td>Dictionary attack.</td>
<td>Customer’s lack of security awareness. System allowing common phrases to be used as passwords.</td>
</tr>
<tr>
<td>Improper initialisation or termination of domains</td>
<td>The customer’s account is suspended but the effect is not reflected on the Auth domain.</td>
<td>No or asynchronous (allowing race condition to happen) propagation of the effect of the change on customer/account status to the Auth domain.</td>
</tr>
<tr>
<td></td>
<td>Improper initialisation of the Auth domain so that every account is associated with the same predefined password.</td>
<td>The domains are improperly initialised causing the system to be in an insecure state.</td>
</tr>
<tr>
<td></td>
<td>Improper initialisation of the Result domain to show authorised status.</td>
<td></td>
</tr>
</tbody>
</table>

For the realisation of external threat represented by the abuse frame diagram in Figure 5, we first investigate the necessary condition on each domain in the large box so that the domain considered can subsequently initiate a sequence of interactions that result in the satisfaction of the anti-

Table 2: Threat realisation, exploited vulnerabilities and attack examples for the right frame diagram in Figure 4.

<table>
<thead>
<tr>
<th>Realisation of the threat</th>
<th>Example</th>
<th>Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interception</td>
<td>Intercepting the customer IDs and passwords sent between the Customer domain and the Customer Identifier machine. The MU then authenticates using the intercepted ID and password.</td>
<td>Communication between the two domains is readable and accessible to an external domain.</td>
</tr>
<tr>
<td></td>
<td>The external domain is a computer on the same network sending false messages to the Result domain, causing it to show a successful authentication and authorisation.</td>
<td>The domains cannot verify or safely assume the authenticity of other domains connected to it.</td>
</tr>
<tr>
<td>Domain/communication fabrication</td>
<td>The external domain is a computer on the same network Inserting false information to the Auth, allowing the MU to be authorised.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The external domain acts as a fake Auth domain and sends false information to the Customer Identifier machine.</td>
<td></td>
</tr>
</tbody>
</table>

In order to address the security vulnerabilities identified, the existing problem descriptions may have to be modified and new security requirements incorporated during the system’s requirements elaboration. The impact
of the revised problem descriptions and security requirements on other aspects of the system’s functionality (e.g., usability and performance) are investigated during a trade-off analysis. The security literature provides a wealth of research in security countermeasures (e.g. [4]) and some examples are shown in the following table.

**Table 3: Some of the suggested new security requirements and revisions to the problem descriptions.**

<table>
<thead>
<tr>
<th>Identified vulnerabilities</th>
<th>Some possible new security requirements and modification to the system context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unlimited tries of password entries</td>
<td>Limit the number of tries for entering passwords;</td>
</tr>
<tr>
<td>Customer’s lack of security awareness. System allowing common phrases to be used as passwords.</td>
<td>Better customer education. Constraining the characteristics of passwords accepted by the system.</td>
</tr>
<tr>
<td>No or asynchronous (allowing race condition to happen) propagation of the effect of the change on customer/account status to the Auth domain.</td>
<td>The system’s updating process shall be <em>atomic</em> with respect to the access of the Auth domain by the Customer Identifier machine;</td>
</tr>
<tr>
<td>The domains are improperly initialised causing the system to be in an insecure state.</td>
<td>The Auth domain shall be initialised to associate each Customer ID with a random password. The Result domain shall be initialised to a predefined state other than <em>authorised</em> and <em>unauthorised</em>.</td>
</tr>
<tr>
<td>The domains cannot verify or safely assume the authenticity of other domains connected to it.</td>
<td>Using dedicated networks to isolate domains from the outside world. Mutual authentication between communicating domains. Verify the authenticity of information with respect to its origin.</td>
</tr>
<tr>
<td>Communication between the two domains is readable and accessible to an external domain.</td>
<td>Use dedicated networks to isolate domains from the outside world. Information shall be transmitted in a form that is not readable to unauthorised domains, <em>e.g.</em>, in cyphertext.</td>
</tr>
</tbody>
</table>

In order to verify that the identified vulnerabilities have been effectively eliminated or reduced and that no new vulnerabilities are introduced, abuse frame analysis is repeated for the modified problem frame descriptions by following the process model outlined in section 2.

5. **CONCLUSIONS AND FUTURE WORK**

This paper presented an overview of our ongoing research in devising a requirements-based technique for analysing security problems. Our technique adopts the problem frame notation, and introduces anti-requirements and abuse frames as two conceptual tools. Through the problem frame notation, abuse frames enable us to express a security threat explicitly by relating undesirable phenomena to a diagrammatic description of the system context. The technique facilitates the structuring of a partial system for early threat analysis when the requirements of the envisioned system are being elaborated. Using an example, we revealed several vulnerabilities relating to the threat of false customer authorisation through the use of abuse frames. We suggest that the incorporation of abuse frame analysis into requirements engineering is a step towards linking security engineering and a general, incremental system’s RE process. However, there are still many open questions that remain for future work. In particular:

- We noted that some security concerns might not be fully addressed until the subproblems are recomposed. A technique is needed for tracing these security concerns throughout the problem decomposition and recomposition process.
- The security community has long investigated in classifying security threats and their realisations. As the problems frames notation is designed to describe patterns of recurring problems, it would be useful for abuse frames to provide a repertoire of security threat patterns by drawing on the knowledge from the security community.
- Abuse frames have focused so far on the structuring of security problems, so research in using abuse frames jointly with other existing techniques for identifying vulnerabilities might be a useful direction to improve the current practices for threat analysis in RE.
- Finally, we are also investigating systematic ways of deriving and revising security requirements once vulnerabilities have been identified.

We intend to explore the above open issues using a large
scale case study.

Acknowledgement. We would like to thank our colleagues Charles Haley, Michael Jackson, and Jonathan Moffett for many insights during discussions. We also gratefully acknowledge the financial support of the Leverhulme Trust.

References

Requirements Elicitation and Analysis Processes for Safety and Security Requirements

Nancy R. Mead
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA. 15213 USA
+1 412 268 5756
nrm@sei.cmu.edu

ABSTRACT
This paper describes a process framework for the elicitation of safety and security requirements and early experience in applying the framework. A larger research project that provides context for the process is briefly described.

Keywords
Computer safety requirements, information security requirements, requirements elicitation, requirements analysis, requirements elicitation process

1 INTRODUCTION
The elicitation and analysis process framework is part of a larger research project, System Quality Requirements Engineering (SQUARE), aimed at improving safety and security requirements engineering practices. The goal for the process framework is to identify and/or develop a recommended process and set of associated techniques for eliciting and analyzing safety and security requirements. Some of the steps to accomplish this goal include the following:

- Identify current processes and techniques for eliciting and analyzing quality requirements.
- Determine the relative strengths and weaknesses of these processes and techniques.
- Determine why these processes and techniques are not being widely used.
- Develop a candidate process and associated set of techniques based on the study of existing processes and techniques.
- Potentially adjust this process based on analysis of SEI experience in security and safety.
- Prototype the use of this process and its techniques.
- Iterate and document the resulting recommended process and set of techniques.

An earlier report has provided a summary of current research in requirements engineering for survivable systems [1]. Earlier papers have also provided research insights [2]. In this paper I describe an elicitation and analysis process that I believe will be useful for safety and security requirements.

2 DRAFT REQUIREMENTS ELICITATION AND ANALYSIS PROCESS
In this process I draw on successful tried and true requirements elicitation methods, as well as newer research methods of the past few years [3, 4, 5]. Significant work has been done recently in the area of risk assessment for survivable systems. A brief discussion of each of the steps follows, and the steps are summarized in Table 1.

Step 1: Candidate definitions for standard security terms, such as confidentiality, integrity, and availability, are presented to stakeholders. The stakeholders have the option to add to the definitions and then to select a final set of definitions for the project. This heads off a debate on the meaning of each term and prevents the confusion that can result when one stakeholder has his or her own idea of what a definition should be and another stakeholder has a different idea.

Step 2: Using business drivers and goals, mission statements, and other artifacts, develop a set of security and survivability goals. This can be a challenge because stakeholders do not necessarily have written business goals, so they may have to be developed first before security goals can be addressed.

Step 3: Select the techniques to be used for requirements elicitation. There is a wide variety of elicitation techniques, and their effectiveness can vary depending on the size of the organization and the formality of their development process. It’s a good idea to pick a few techniques and then narrow them down before proceeding with the next steps.

Step 4: Develop the artifacts necessary to support the elicitation technique. Use cases and misuse cases are popular tools to support security requirements elicitation. Threat models and attack patterns also provide useful input. For reuse, templates can be introduced at this point in the process. This can be a very time-consuming activity, as
many organizations don’t have the needed artifacts for the “normal” case, such as use cases and architecture diagrams.

Step 5: Elicit the actual requirements using the techniques selected in step 4 and the support of the artifacts developed in step 4. One difficult area is the phrasing of the requirements so that they represent real, testable requirements rather than architecture/design constraints or lofty goals that cannot be met.

Step 6: Categorize the requirements and assess whether they are really requirements or other kinds of constraints, as noted in step 5.

Step 7: Perform a risk assessment activity. There is a wide variety of choices, including traditional software risk assessment, OCTAVE [3], or other methods [4, 5].

Step 8: Select a method for prioritizing requirements and go through the prioritization process. Priorities could be based on the likelihood that the risk will become a reality, cost/benefit analysis, areas of particular concern for the stakeholders, etc.

Step 9: Inspect the requirements using a standard inspection or review process to ensure that they are consistent, complete, testable, etc.

This summer’s work, described below, shows that the draft process can be applied to systems concerned with security properties, and it may be applicable to systems with safety properties as well.

3 EXPERIENCE WITH THE DRAFT PROCESS

I am in the early stages of applying the process with real clients. So far, I am finding that organizations are very interested in the work that I am doing and they want to improve their processes. However, they are very busy with development work and have little time for process improvement. It is also the case that I have to start with the basics, such as defining terms, developing architectural diagrams, and helping to identify business goals and mission statements. Small to mid-size companies typically do not have the infrastructure to just add a new process and run with it. Thus, I am spending significant time to get the organizations to the point where they can benefit from new processes and tools. It remains to be seen whether the ideal time to apply this process is after a system architecture exists or prior to architecture development. Initial findings suggest that an architecture needs to be in place at some level in order for this process to be successful.

This summer, I supervised seven graduate students in applying the process and developing some rudimentary tools. Five students worked as a team with a client, under my supervision. Two student interns worked on extending the process descriptions and developing some basic Web-based tools, also under my supervision.

A client organization we were working with initially had to withdraw due to the pressure of tight deadlines, a problem that is all too common in today’s business environment. The current client organization has been very responsive and supportive of the work, and has stayed with us through completion of the pilot exercise. The current client is also willing to continue in-depth analysis of some of the steps in the fall, with a different student group. Some of the current students may continue with independent studies or further internships rather than the team project. This client exercise focused on security requirements.

Results and Lessons Learned from the Summer Experience

The student project was very successful in providing feedback on the method and in providing the client with suggested improvements in their information security posture.

Here’s a synopsis of the findings:

The client, a small start-up company, was in the process of expanding the market for an existing product to include users who might be installing the product on networks with Internet access. The current users typically install the product on isolated networks, where security concerns are minimal. Hence the client was motivated to review and improve their security posture.

As a first step, I described the research project to the client, and they were interested in working with the students. I requested copies of existing documentation, which were provided to the students, and arranged a subsequent student/client meeting.

It quickly became clear that the client organization had user documentation but little, if any, architecture or requirements documentation. This is a fairly typical state of affairs for many organizations. Hence, the students had to spend a fair amount of time eliciting and documenting the architecture, business goals, and other requirements. Since there was a large team of students, many activities took place in parallel, rather than sequentially.

There is not sufficient definition of the process itself. Each process step needs to be broken down into substeps, with explanatory notes. This is an exercise for me, with review by research colleagues. The students needed a lot of guidance because the steps were not sufficiently detailed or self-explanatory.

For step 1, a set of candidate definitions was developed. Definitions were recommended to the client, and the client picked a final set of definitions to be used for the project. A similar exercise took place in identifying the architecture goals, in this case associated with security.

Most of the time this summer was focused on steps 3 through 5, which were executed in an iterative process. Step 3, selecting elicitation techniques, was not really done.
The students just iterated with the client to identify use and misuse cases. This occurred in part because there was one individual who served as the primary client, and a second who provided higher level review. As a result, some of the difficulties that occur with large groups of stakeholders did not take place, and surveys and other sophisticated techniques were not needed. Additionally, it may be the case that stakeholder selection is important at each step, and different stakeholders may need to be involved at different steps in the process. This is an area that could use more study in the fall.

One student subteam developed a set of use cases based on working on the actual client product at the client location. Another student subteam developed a set of misuse cases. Much time was spent iterating on the misuse cases to decide the best way to document them. The use cases and misuse cases were ultimately documented in both tabular form and in Visio. In addition to the misuse cases, some attack trees were developed to test completeness of the misuse cases. Unfortunately, there was insufficient time to develop a complete set of attack trees, so this activity will be picked up in the fall. It has become clear that a threat model, based on attack trees or some other method, is needed in order to develop a good set of requirements [6].

Once the use and misuse cases were done, architectural and policy recommendations for improving security were developed. These were traced to the misuse cases. From these recommendations, the students developed a set of implementation choices for the client. At this point, the students were ready to abstract these recommendations into a set of requirements. Ideally we wanted to trace the security goals to requirements, which in turn would trace to architectural recommendations and implementation choices. This turned out to be a difficult task, in part because the students had no experience with requirements engineering, and in part because we had difficulty writing a security requirement that is free of implementation details. We believe that development of the threat model will help with this.

One of the students did a risk assessment (step 7), but this was relatively independent of the other activities. The risk assessment would also help in defining requirements, so this is another loop that needs to be closed.

In parallel with this, the students attempted to prioritize the misuse cases, with input from the client, do a cost/benefit analysis, and use the cost/benefit analysis to help with the prioritization. Initially the students and the client prioritized the misuse cases as high/medium/low (there were no cases categorized as low in the end). The high priority cases were further addressed with the cost/benefit analysis. Several prioritization and analysis models were considered before arriving at the final selection.

In the end, there was insufficient time for the inspection process. This will also be picked up in the fall.

Considering that there were seven students working for 12 weeks at an average of 30 hours a week, it’s clear that the process is quite time consuming. However, some of the time was spent in learning about the requirements area and also in documenting the current goals and architecture on behalf of the client.

**RHAS Workshop**

The students have completed their work for the summer, and there is good feedback on the usefulness of the process, as well as further documentation of the process. The students produced two large reports – one report for the client with the actual results of the process, and another report providing feedback on the process itself.

I am hoping that the workshop participants will provide professional feedback on the content of the process, whether it applies equally to safety and security and, if not, what the differences are. I am also hopeful that the workshop participants will provide advice on how to identify appropriate clients for applying the draft process (that is, what would be the characteristics of clients who would be able to apply new processes without having to start with fundamentals).

### 4 FUTURE WORK

My intent is to refine the process and gain more experience with a variety of clients. Toward the end of the summer we uncovered a number of new references and practices that would help to identify better security requirements [7]. These will be factored into our thinking. I will continue to track the literature for new approaches to safety and security requirements elicitation and analysis. Research in this area is needed to improve the safety and security of critical systems.

### REFERENCES


<table>
<thead>
<tr>
<th>Step No.</th>
<th>Step</th>
<th>Input</th>
<th>Techniques</th>
<th>Participants</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Agree on definitions</td>
<td>Candidate definitions from IEEE and other standards</td>
<td>Structured interviews, focus group</td>
<td>Stakeholders, requirements team</td>
<td>Agreed-to definitions</td>
</tr>
<tr>
<td>2</td>
<td>Identify safety and security goals</td>
<td>Definitions, candidate goals, business drivers, policies and procedures, examples</td>
<td>Facilitated work session, surveys, interviews</td>
<td>Stakeholders, requirements engineer</td>
<td>Goals</td>
</tr>
<tr>
<td>3</td>
<td>Select elicitation techniques</td>
<td>Goals, definitions, candidate techniques, expertise of stakeholders, organizational style, culture, level of safety and security needed, cost benefit analysis, etc.</td>
<td>Work session</td>
<td>Requirements engineer</td>
<td>Selected elicitation techniques</td>
</tr>
<tr>
<td>4</td>
<td>Develop artifacts to support elicitation technique</td>
<td>Selected techniques, potential artifacts (e.g., scenarios, misuse cases, templates, forms)</td>
<td>Work session</td>
<td>Requirements engineer</td>
<td>Needed artifacts: scenarios, misuse cases, models, templates, forms</td>
</tr>
<tr>
<td>5</td>
<td>Elicit safety and security requirements</td>
<td>Artifacts, selected techniques</td>
<td>Joint Application Design (JAD), interviews, surveys, model-based analysis, safety analysis, checklists, lists of reusable requirements types, document reviews</td>
<td>Stakeholders facilitated by requirements engineer</td>
<td>Initial cut at safety and security requirements</td>
</tr>
<tr>
<td>6</td>
<td>Categorize requirements as to level (system, software, etc.) and whether they are requirements or other kinds of constraints</td>
<td>Initial requirements, architecture</td>
<td>Work session using a standard set of categories</td>
<td>Requirements engineer, other specialists as needed</td>
<td>Categorized requirements</td>
</tr>
<tr>
<td>7</td>
<td>Perform risk assessment</td>
<td>Categorized requirements, target operational environment</td>
<td>Risk assessment method, analysis of anticipated risk against organizational risk tolerance, including hazard/threat analysis (OCTAVE, Shawn Butler, Martin Feather)</td>
<td>Requirements engineer, risk expert, stakeholders</td>
<td>Risk assessment results, added mitigation requirements to bring exposure into acceptable level</td>
</tr>
<tr>
<td>8</td>
<td>Prioritize requirements</td>
<td>Categorized requirements and risk assessment results</td>
<td>Prioritization methods such as Triage, Win-Win, etc.</td>
<td>Stakeholders facilitated by requirements engineer</td>
<td>Prioritized requirements</td>
</tr>
<tr>
<td>9</td>
<td>Requirements inspection</td>
<td>Prioritized requirements, candidate formal inspection technique</td>
<td>Inspection method such as Fagan, peer reviews, etc.</td>
<td>Inspection team</td>
<td>Initial selected requirements, documentation of decision making process and rationale</td>
</tr>
</tbody>
</table>
From System Safety Analysis to Software Specification

Elena Troubitsyna
Department of Computer Science
Åbo Akademi University
Lemminkaisenkatu 14 A
20520, Turku, Finland
+358 2 215 4086
Elena.Troubitsyna@abo.fi

ABSTRACT
In this paper we demonstrate how to derive software requirements from system safety analysis and capture them in a formal specification. We propose an integral approach for incorporating results of Fault Tree Analysis (FTA) and Failure Mode and Effect Analysis (FMEA) into the formal specification in the B Method. In our approach statecharts facilitate construction of the control system and serve as a basis for structuring and integrating results of FTA and FMEA. The use of statecharts as a communication media between safety and software engineers assists the process of requirements discovery. The use of formal development technique ensures correctness of implementing the requirements. The approach is illustrated by excerpts from the development of realistic industrial system – the liquid handling workstation Fillwell™.

Keywords
Safety, FTA, FMEA, statecharts, formal specification, B

1 INTRODUCTION
In this paper we aim at creating an integral method for formal software development which would naturally incorporate requirements from safety analysis [10,17] into the process of software construction. Currently the design environment of dependable systems is fragmented [9]. Indeed, even though safety analysis supplies critical requirements to be imposed on controlling software, safety analysis is yet not well integrated with software development process [11]. On the one hand, there is a traditional culture of isolating software development from hardware development and safety analysis. On the other hand, there is a lack of methods inter-relating dependability and software modeling.

The approach presented in this paper is a reflection on our experience gained while developing the liquid handling workstation Fillwell™ [3]. The development was undertaken as a part of the EU project MATISSE [12]. Within the project we carried out a formal development of controlling software for the workstation in the B Method [1]. We experienced a semantic gap between the results of safety analysis and software requirements. In this paper we propose to bridge the gap via statecharts modeling. The visual nature of statecharts provides a fruitful environment for the interdisciplinary communication and allows for a smooth transition from safety analysis to software requirements and then to the formal specifications in B.

In this paper we describe our approach to integrating results of FTA and FMEA [10,17] into the formal system specification. We demonstrate how not only safety invariant but also the mechanisms for error detection and recovery [2] can be derived from safety analysis and specified formally. We illustrate our approach by the corresponding excerpts from the development of Fillwell.

2 DISCOVERING SOFTWARE REQUIREMENTS WITH FTA
Safety analysis is devoted to identifying hazards, determining their causes, and deciding on their elimination and mitigation [10,17]. In this paper we focus on integration of two techniques – Fault Tree Analysis (FTA) and Failure Mode and Effect Analysis (FMEA) into the formal process of software development.

Fault Tree Analysis
FTA is a deductive safety analysis technique [10,17]. It is a top-down approach applied during all design stages. Preliminary hazard identification provides information about hazardous situations in functioning of the system. This information is taken as an input for the FTA. The result of the FTA is an identification of combinations of components states that result in hazards. Each fault tree has a root representing a hazardous situation. The tree traces the system to the lower component level to reveal the possible causes of failures.

To illustrate FTA let us consider an example – a simplified version of Fillwell [3] – a liquid handling workstation which was developed as a case study within the project MATISSE [12]. The workstation belongs to the class of products for drug discovery and bioresearch. The system consists of an operating head dispensing liquid substances into and aspirating them from micro-plates placed on a processing table. A gantry moves the operating head with high precision and speed from one plate to another in XYZ-directions. The head provides precise dispensing into high-
density micro-plates. The schematic representation of the workstation is given in Fig.1.

![Schematic representation of Fillwell](image)

Figure 1. Schematic representation of Fillwell

The system’s components XCOMP, YCOMP, and ZCOMP are responsible for moving the gantry and the operating head in X,Y,Z-directions correspondingly. The component PUMP aspires and dispenses liquid by means of a peristaltic pump. The workstation is used to perform various pharmaceutical and bioresearch experiments. To perform an experiment a user defines a sequence of high-level commands – a protocol. Essentially a protocol consists of aspirating and dispensing liquid from one plate to another. The high-level commands are decomposed into sequences of lower-level commands activating system components in a certain order or in parallel.

The major safety and reliability requirement imposed on the system is to perform experiments correctly. Hence any situation which can potentially lead to an incorrect execution of the experiments is considered to be hazardous. For instance, among such situations is “damage of the equipment in the course of the experiment”. It includes the states of the system in which safety operating boundaries are breached, the operating head collides with the table or the plates etc. In Fig.2 we present an excerpt from a high-level fault tree which traces the causes of the hazardous situation “head collides with plate” to the states of the system’s components.

Let us note that the fault tree provides a logical representation of the hazardous system state in terms of the component states:

\[(<\text{XCOMP active}> \lor <\text{YCOMP active}>) \land <\text{ZCOMP active}>\] (1)

Our experience shows that the fault trees for the component’s states contributing to a hazard can be constructed on the basis of the pattern represented in Fig. 3.

![Fault tree diagram](image)

Figure 2. An example of a fault tree

Therefore we conclude that most of the software-related problems can be tackled by a method for a precise and unambiguous translation of results of safety analysis into formal software specification.

Often this problem is handled by giving fault trees formal semantics and then formulating safety invariant, which controlling software should preserve (e.g., [5]). However, this approach has a number of drawbacks. Firstly, not all the leaves of fault trees are related to software requirements (e.g., hardware failures should be addressed by introducing hardware redundancy), so an application of the approach is not always straightforward. Secondly, such an approach does not support inter-disciplinary communication because formal notation can be rather cumbersome to perceive for safety engineers. Finally, with such an approach it is rather difficult to correlate changes in software specification or...
Figure 3. Fault tree of undesirable component state system structure with safety. In this paper we propose an approach to tackling these problems.

From fault trees to software requirements via statecharts

It is widely accepted that visual techniques facilitate development of complex systems as well as interdisciplinary communication [4]. Our approach is based on using statecharts as means for integrating safety and functional requirements and translating them into a formal software specification. Statecharts [6] is a visual formalism which can be seen as a generalization of the conventional finite state automata. Statecharts form a part of a popular object-oriented modeling technique – Unified Modeling Language (UML) and are widely used nowadays. The formalism supports such features as hierarchy, concurrency and broadcast communication between system components.

Statecharts describe system’s behavior as an evolution from one state into another upon arrival of an event. To illustrate statecharts modeling let us return to our example – the Fillwell workstation. The statecharts representation of the correct behavior of the system is given in Fig.4.

The system’s components XCOMP, YCOMP, and ZCOMP and PUMP are represented as parallel activities. The parallel activities are rendered by splitting the system state with a dashed line. As a result each sub-state represents the behavior of the corresponding component. In general, the behavior of all the components follows the same pattern: while a component is idle (not engaged in executing any command) it can be activated. As a response to its activation a component executes a standard control loop (i.e., reading sensors and assigning actuators) to perform a requested command.

For example, let us analyze the behavior of XCOMP when it receives a request to move the operating head from its current position to the position XP1. Assume that currently the operating head is in its XHome position. Placing the request is modeled by the event XMove. The request activates the component XCOMP. The active (engaged) state of the component is modeled by the state XEngHome. Upon leaving the position XHome the operating head switches off the home sensor XHS which is modeled by the event XHSOFF. The arrival of the operating head at the position XP1 switches on the sensor XSP1 which is depicted by the event XSP1ON. Since the requested destination is XP1, the condition \[ x_{dest} = XP1 \] is satisfied and the component XCOMP arrives at the state XIdleP1. In this state XCOMP becomes inactive again. The behavior of the other components is modeled in the similar way. Due to lack of space we omit their detailed discussion.

Next we demonstrate how statecharts can assist in deriving safety invariant from the FTA. Let us note that the behavior of the entire system is represented by the statecharts depicting system’s components functioning in parallel. At each instance of time each component is in a certain state. Hence, at each instance of time we can receive a “snapshot” of all components states. Let us also observe that lower level leaves of fault tree correspond to certain states of components. Hence, by matching these leaves with the corresponding component states we connect the fault tree with the statecharts model of the system. Furthermore, minimal cut sets of a fault tree logically describe the unsafe combinations of component states which lead to the hazard.

We illustrate this approach by expressing the hazard “damage of the equipment in the course of the experiment” in terms of component states. We extend the fault tree given in Fig.2. to incorporate the results of statecharts modeling. In Fig.5 we show an excerpt from the resultant fault tree. It represents the active state of XCOMP as a fault tree of its states. The hazard expressed on a high level (1) corresponds to the following logical expression over component states:
Fig. 4. Excerpt from Fillwell statecharts

Hazard == (XCOMP=XEngHome ∨ XCOMP=XMovingP1 ∨ XCOMP=XEngP1 ∨ XCOMP=XMovingP1P2 ∨ XCOMP=XEngP2 ∨ XCOMP=XMovingP2P3 ∨ XCOMP=XEngP3) ∧ (ZCOMP=ZEngHome ∨ ZCOMP=ZMovingP1 ∨ ZCOMP=ZEngP1 ∨ ZCOMP=ZMovingP1P2 ∨ ZCOMP=ZEngP2 ∨ ZCOMP=ZMovingP2P3 ∨ ZCOMP=ZEngP3)

By negating this expression we obtain the safety condition which the controlling software should preserve to avoid the damage of the equipment:

Safety_cond= ¬Hazard

In the similar way the other hazards (e.g., “activating the pump while the operating head is moving”) are traced to the components states. The overall safety invariant is formed as a conjunction over the logical expressions defining safety conditions to avoid the corresponding hazards.

As a result of integrating statecharts modeling and fault tree analysis we obtained a simple and unambiguous way to translate safety requirements into safety invariant. However, an application of the FTA alone is yet insufficient for completeness of safety requirements. Note that while constructing a fault tree we used the fact that, e.g., a component has failed and is in a certain state but we did not identify the means for error detection. Below we propose an approach to structuring and formalizing the results of FMEA which facilitate extracting the requirements for error detection and recovery.

3 EXTRACTING FAULT TOLERANCE REQUIREMENTS FROM FMEA

FMEA [10,17] is an inductive analysis method, which allows us to systematically study the causes of components faults, their effects and means to cope with these faults. FMEA is used to assess the effects of each failure mode of a component on various functions of the system as well as to identify the failure modes significantly affecting dependability of the system. FMEA supplies the information about failure modes of the individual components into the FTA. FTA and FMEA are often conducted together to compliment each other.

Even though FMEA should provide us with important requirements describing fault tolerance mechanisms often these requirements are rather difficult to extract. A major reason of this is a hardware-oriented style of FMEA resulting in ignoring the software aspect. Indeed, while safety engineers usually give a precise and detailed description of hardware-based fault tolerance mechanisms, they often describe software-based fault tolerance by a phrase like “modify software to detect error and implement error recovery” [17]. Obviously, such a description is too vague to result in precise software requirements.

In this paper we propose to tackle this problem by “enforcing” a structured way to describe software-based fault tolerance mechanisms on the basis of statecharts. In Fig.6 we present our proposal for integrating such a structured description into a traditional FMEA representation. Further description of the approach can be found in [18]. The FMEA table is extended with the explicit description of error detection mechanisms and structured description of the software implementation of remedial actions. To exemplify the proposed approach in
Fig. 7 we present an excerpt from an analysis of the failure mode "stuck at zero" of the XP1 sensor. The proposed extension enables a straightforward process of deriving requirements for software-based fault tolerance. Next we demonstrate how the requirements obtained from FTA and FMEA can be integrated with functional requirements in a formal specification.

4 INTEGRATING SAFETY ANALYSIS AND FORMAL SPECIFICATION

Correctness of controlling software is critical for system dependability [9]. Traditionally software correctness is ensured by an application of formal development methods. While selecting a formal specification framework we considered such criteria as simplicity of notation, availability of an automatic tool supporting development and verification and finally, possibility to automate translation from statecharts to the formal specification. As a result the B Method has been chosen.

Specifying in B

The B Method (hereinafter referred to as B) is an approach for the industrial development of highly dependable software [1]. The method has been successfully used in the development of several complex real-life applications [12]. The tool support available for B provides us with the assistance for the entire development process. For instance, Atelier B [16], one of the tools supporting the B Method, has facilities for automatic verification and code generation as well as documentation, project management and prototyping.

The development methodology adopted by B is based on stepwise refinement. While developing a system by refinement we start from an abstract formal specification and transform it into an implementable program by a number of correctness preserving steps, called refinements.

A formal specification is a mathematical definition of requirements imposed on a system. In B a specification is represented by a set of modules, called Abstract Machines.
The special case of a parallel composition is a multiple assignment which is denoted as \( x,y := e_1,e_2 \).

Modelling control system in B
The process of translating statecharts representation of system’s behaviour into B is rather straightforward. In the abstract machine we represent concurrent components, superstates, states and events of statecharts as corresponding state variables. Initial states in statecharts correspond to initialization of the appropriate variables of the B machine. Transitions between states are modeled by the operations of abstract machine. In general, the transition from the state \( S \) in the superstate \( SS \) to the state \( S' \) in the superstate \( SS' \) upon arrival of the event \( E \) can be specified by an operation of the following form:

\[
\text{OP} = \text{SELECT EVENT}=E \& \text{SUPERSTATE}=SS \& \text{STATE}=S \\
\text{THEN SUPERSTATE}:=SS' \| \text{STATE}:=S' \text{ END}
\]
Conditions are expressed as predicates over state variables. An operation of the form

\[
\text{OP} = \text{SELECT Event=E & Superstate=SS & state=S} \\
\quad \text{THEN IF cond1} \\
\quad \quad \text{THEN Superstate=SS'} \text{|| state=S'} \\
\quad \quad \text{ELSEIF cond2} \\
\quad \quad \text{THEN Superstate=SS'' || state=S''} \\
\quad \text{END} \\
\quad \text{END}
\]

models a transition from the state \( S \) in the superstate \( SS \) to the state \( S' \) in the superstate \( SS' \) upon arrival of the event \( E \) if the condition \( \text{cond1} \) holds and to the state \( S'' \) in the superstate \( SS'' \) upon arrival of the event \( E \) if the condition \( \text{cond2} \) holds. An elaborated description of translating statecharts into B can be found in [13].

Usually a control system is a reactive system with two main entities: a plant and a controller. The overall behavior of the system is an alternation between the events modeling plant evolution and controller reaction. As a result of the initialization, the plant’s operation becomes enabled. Once completed, the plant enables the controller. The controller monitors the behavior of the plant and adjusts it to provide intended functionality and maintain safety. The controller is specified as a composition of operations modeling

- routine control, i.e., reaction of the controller on the normal (fault-free) and safe behaviour of the plant
- error detection, i.e., the operations which are enabled when erroneous state of the system is detected
- remedial actions, i.e., the operations which specify the behaviour of the controller while recovering the system from errors

The general structure of a control system which we propose is given in machine ControlSystem

\[
\text{MACHINE} \\
\text{ControlSystem} \\
\text{VARIABLES} \\
\quad \text{flag, state\_variables} \\
\text{INVARIANT} \\
\quad \text{flag : \{pl,contr\} & safe} \\
\text{INITIALIZATION} \\
\quad \text{flag := pl ...} \\
\text{OPERATIONS} \\
\quad \text{Plant = SELECT flag=pl THEN generate\_stimuli ||} \\
\quad \quad \quad \text{flag := cont END;} \\
\quad \text{Control = SELECT flag=contr & safe & plant\_stimulus} \\
\quad \quad \quad \text{THEN control\_action || flag := pl END;} \\
\quad \text{Detection = SELECT flag=contr & error\_detected} \\
\quad \quad \quad \text{THEN Initiate remedial\_actions END ;} \\
\quad \text{Remedy = SELECT flag=contr & plant\_stimulus} \\
\quad \quad \quad \text{THEN remedial\_action || flag := pl END}
\]

The safety invariant \( \text{safe} \) is obtained by merging via conjunction safety conditions obtained as a result of FTA. The plant is specified as a non-deterministic choice of events. Such a specification allows us to model not only normal behaviour of the plant but also errors which manifest themselves as aberrant events or timeouts. The routine control is specified by the set of operations of the general form Control. The operation Control is enabled if the plant is safe and has generated stimulus corresponding to the fault-free behaviour. In contrast the operation Detection becomes enabled if one of the error detection mechanisms has been trigged by the stimulus obtained from the plant. The predicates \( \text{error\_detected} \) in the guards of Detection operations are obtained from the extended FMEA description of the error detection mechanisms. In our approach the predicates defining detection conditions given as aberrant events have identical form:

\[
\text{Event=}\text{D\_event & state=}\text{D\_state}
\]

obtained by a straightforward translation of the information given in “Detection” row of extended FMEA. We model error detection by timeout in the similar way:

\[
\text{Timer=ON & t-TimeStamp} > \text{Constraint}
\]

where \( t \) is a variable modeling time.

However, this requires activation/deactivation of timer by the normal control operations which are enabled when the events \( \text{A\_event} \) and \( \text{DA\_event} \) arrive. The activation of timer \( \text{Timer := ON} \) and deactivation of timer \( \text{Timer := OFF} \) are added to these actions as simultaneously executed statements.

Finally, the remedial actions are of the form \( \text{Remedy} \). They are activated upon detection of corresponding errors. The operations specify the reaction of the controller on the stimuli of faulty plant in the process of error recovery. Upon successful completion of error recovery the controller resumes normal control, i.e., the operations \( \text{Control} \) become enabled again. Upon failure of error recovery the remedial actions shut down the system.

**Fillwell: Formal B Specification**

To illustrate formal modeling of discrete control systems in B, in Fig. 8 we present an excerpt from the specification of Fillwell. The system is specified according to the general specification form ControlSystem. The plant is modeled by the operation \( \text{XPlant} \). The operations \( \text{XC}_i:1..N \) specify routine control operations to be provided by the component \( \text{XCOMP} \) in response to stimuli of fault-free plant. We assume that the reaction of the controller takes negligible amount of time so the controller can react properly on changes of the plant state.

The operation \( \text{XP1SF\_D} \) specifies detection of the error “\( \text{XSP1 stuck at zero} \)” . Observe that the specification is obtained by the straightforward translation of the description of the detection mechanism. The corresponding remedial actions \( \text{XP1SF\_R1..XP1SF\_R4} \) are obtained on
**MACHINE**

XFillwell

**SETS**

XIEVENTS : {XHSOFF,XP1SON,...};
XEEVENTS: {XMove};
XSTATE : {XIdleHome,XEngHome, XMovingHP1, XIdleP1,...};
XDEST : {XHome, XP1,XP2,XP3};

**VARIABLES**

fl, xe, xst, xdest, fdest...

**DEFINITIONS**

safe = safety_cond1 & safety_cond2 &

**INVARIANT**

fl : {cont,pl} & xe : XIEVENTS / XEEVENTS &

xst : XSTATES & xdest, fdest : XDEST & safe

**OPERATIONS**

XPlant = SELECT fl= pl THEN

CHOICE xe : XIEVENTS

OR xe := XMove || xdest := XDEST END || fl := cont

XC_1 = SELECT fl= cont & xe:= XMove & xst= XIdleH && safe

THEN IF xdest = XHome THEN skip

ELSE xst := XEngHome || fdest := xdest END

|| fl := pl END

XC_2 = SELECT fl=cont & xe= XHSOFF & xst= XEngHome &

safe

THEN xst := XMovingHP1 || fl := pl END

XC_3 = SELECT fl=cont & xe=XP1SON & xst=MovingHP1 &

safe

THEN IF xdest = XP1 THEN xst:= XIdleP1

ELSE xst := XEngP1 END || fl := pl END

XP1SF_D = SELECT fl=cont & xe=XP2SON & xst=XMovingHP1

THEN xst := XP1SF_DET || xe,xdest:= XMove,XP1||

PauseY END

XP1SF_R1 = SELECT fl=cont & xe=XMove & xst= XP1SF_DET

THEN xst := XP1SF_Eng || fl := pl END

... XP1SF_R3= SELECT fl=cont & xe=XP1SON &

xst= XP1SF_MovingP2P1

THEN xst := XIdleP1 || xe,xdest := XMove,fdest END

XP1SF_R4= SELECT fl=cont & xe=XP1SON &

xst= XP1SF_MovingP2P1 THEN xst:=XStopped END

END

Fig.8. Excerpt from formal specification of Fillwell.

The basis of statecharts representation of the recovery action (we omitted the presentation of these statecharts – they merely define course of events in the recovery action). The recovery action XP1SF_R4 is executed in case of failure of error recovery – it brings the component in a non-operational state XStopped.

The initial formal specification contains nondeterminism and abstract data types. They are replaced by implementable constructs in the process of system refinement. The final refinement step decomposes the system into a plant and controller, i.e., allows us to arrive at the specification of the controlling software as such. Finally, executable code is generated.

The process of obtaining B specifications from statecharts is facilitated by the U2B tool [15]. Moreover, we are developing a tool which supports FTA and extended FMEA and integrates their results into the B specifications as described above.

5 CONCLUSIONS

In this paper we presented an approach to integrating safety analysis with formal development. The approach enables a smooth transition from reasoning about safety to specifying controlling software. Our specifications are formal and can be (automatically) transformed into executable programs through a succession of refinement steps. In this paper we omitted a detailed description of refinement process (it can be found, e.g., in [7,8,14]). Instead we focused on capturing requirements supplied by FTA and FMEA in the formal specifications. The use of statecharts allowed us to interrelate software and safety modeling and significantly simplified derivation of safety-related requirements. The various stages of the approach have been automated and allowed us to scale up formal development.

In our future work it would be interesting to integrate the other safety techniques with formal modeling and eventually defragment the design environment of dependable systems.

References